Erciyes Üniversitesi
Fen Bilimleri Enstitüsü Dergisi
Cilt 41, Sayı 2, 2025

Erciyes University
Journal of Institue Of Science and Technology
Volume 41, Issue 2, 2025

Numerical Solutions for Differential Equations Using Matlab

Bengü Çına [1*]

[1*] Cumhuriyet University, Zara Veysel Dursun School of Applied Science, SİVAS

KeyWords
Ode45,
Runge-Kutta,
Dormand-Prince (4,5),
Matlab.

Abstract: The role of differential equations in modelling a range of phenomena across disciplines, including physics, engineering, biology and economics, is of great significance. Despite many differential equations can be solved analytically, others present a challenge in this regard. MATLAB can be employed to facilitate the resolution of these intricate equations. The aim of this study is to obtain numerical solutions of differential equations that have no analytical solutions or whose solutions are complex using MATLAB and to analyse the graphs of these solutions. In this way, we aim to gain a deeper understanding of the dynamic behaviour of the equations and their solution ranges. For this purpose, the ode45 function and the Runge-Kutta method will be mostly used. In addition, the study includes several definitions and theorems that support the theoretical background and provide a framework for the numerical approaches applied.

Diferansiyel Denklemler İçin Matlab Kullanarak Nümerik Çözümler

Anahtar Kelimeler
Ode45,
Runge-Kutta Yöntemi,
Dormand-Prince (4,5),
Matlab.

Öz: Diferansiyel denklemlerin fizik, mühendislik, biyoloji ve ekonomi gibi disiplinlerdeki bir dizi olgunun modellenmesindeki rolü büyük önem taşımaktadır. Birçok diferansiyel denklem analitik olarak çözülebilmesine rağmen, diğerleri bu konuda bir zorluk teşkil etmektedir. MATLAB, bu karmaşık denklemlerin çözümünü kolaylaştırmak için kullanılabilir. Bu çalışmanın amacı, analitik çözümü olmayan veya çözümü karmaşık olan diferansiyel denklemlerin MATLAB kullanılarak sayısal çözümlerini elde etmek ve bu çözümlerin grafiklerini analiz etmektir. Bu sayede denklemlerin dinamik davranışları ve çözüm aralıkları hakkında daha derin bir anlayış kazanmayı hedefliyoruz. Bu amaçla çoğunlukla ode45 fonksiyonu ve Runge-Kutta yöntemi kullanılacaktır.

1.Introduction

The objective of numerical analysis is to overcome complex numerical challenges by employing only the fundamental operations of arithmetic. This entails the formulation and assessment of techniques for the calculation of numerical outcomes from specified data. The computational methods are called algorithms. An algorithm is a set of instructions that defines a sequence of operations to be performed by a computer. At each stage of the operation, the instructions tell the computer exactly what to do. Applied numerical methods therefore often focus on practical applications and real-world problem solving. They are designed to provide fast and accurate solutions to complex mathematical problems that can't be solved analytically. This includes a range of techniques such as some difference, some element and spectral lines, which are used in many different fields including engineering, physics, finance and data science. Moreover, the success of these methods is often measured by their computational speed, stability and accuracy in reproducing the original mathematical models [1]. This means that not only do practitioners use these algorithms, but they also constantly modify them to make them work better in different situations. The field of numerical methods for solving differential equations is one that has been the subject of sustained academic interest for many years.

The use of differential equations is a fundamental aspect of mathematical modelling, with applications across a diverse range of scientific, engineering, economic, mathematical, physical, aeronautical, astronomical, dynamical, biological, chemical, medical, environmental, social, banking and other disciplines. Despite the existence of numerous analytical techniques for solving differential equations, there remains a significant number that cannot be solved analytically. This implies that the solution cannot be expressed as the sum of a finite number of elementary functions, including but not limited to polynomials, exponentials, trigonometric and hyperbolic functions. In the case of simple differential equations, it is possible to find closed-form solutions[2]. However, many differential equations that arise in applications are so complex that it is not always feasible to have solution formulas. Alternatively, if a solution formula is available, it may involve integrals that can only be calculated using a numerical quadrature formula. In either case, numerical methods provide a powerful alternative tool for solving the differential equations under the prescribed initial condition or conditions. B. Dennis [3] studied on the basic and commonly used numerical and analytical methods of solving ordinary differential equations. Ş. Yüzbaşı et. al.[4] gave a numerical method for solving systems of higher order linear functional differential equations using MATLAB. M. Saqib et al.[5] concentrated on dynamical behavior of nonlinear coupled reaction-diffusion model in Matlab. D. Gopal et. al. [6] conducted a numerical investigation of a higher-order chemical reaction using MATLAB. The governing equations for the fluid flow are coupled and involve nonlinear partial derivatives. The impact of electric and magnetic fields on a nanofluid with viscous dissipation in the presence of a higher-order chemical reaction, with a focus on the conservation of momentum and energy, represents a novel aspect of the problem. A. Thabet et. al. [7] studied on numerical solutions and made significant contributions to the class of time–space fractional partial differential equation using matlab. B. W. Ong and R. J. Spiteri [8] concantrated on deferred correction (DC) methods for ordinary differential equations. They use the terminology "DC method" to generally refer to the process of refining the numerical solution to an ODE by iteration. DC methods have been extensively applied to IVPs [9-14] and BVPs [15-17] for ODEs, initial-boundary value problems for PDEs [15,18,19] differential-algebraic equations [19,20], and eigenvalue problems [19,21]. More recently, Reuter B., et al. [22] aimed to design a research-oriented, yet computationally efficient software tool for solving partial differential equations (PDEs). In the study, various discontinuous Galerkin (DG) methods were used for spatial discretisation, while different explicit, implicit or semi-implicit Runge-Kutta pattern were employed for the time step. The resolution of differential equations via numerical methods has long constituted a subject of rigorous academic study, wherein a plethora of techniques have been developed to address the distinctive challenges posed by disparate problem types [23-25]. In addition, readers are advised to consult references [30,31] for further examples.

This paper presents several different examples of numerical solutions for differential equations that lack either an analytical solution or a behavior that can be described by a complex model, and includes theorems that support the analysis and validity of these solutions. Matlab is employed as the principal tool for numerical computations. The graphs produced are analysed to gain insight into the behaviour of the differential equations and their potential solutions. The ode45 function, based on the Runge-Kutta method, is utilized to facilitate rapid and efficient numerical integration.

2. Materials and Methods

Definition 2.1 Runge–Kutta Method [26]

Consider first-order initial-value problem:

$$y' = f(x, y), \quad a \le x \le b \qquad (1)$$
$$y(a) = y_0$$

To derive the Runge–Kutta method, we divide the interval $[a, b]$ into N subintervals as $[x_n, x_{n+1}]$ $(n = 0,1, \dots, N-1)$ integrating $y' = f(x, y)$ over $[x_n, x_{n+1}]$ and utilizing the mean value theorem for integrals, we obtain

$$y(x_{n+1}) - y(n) = \int_{x_n}^{x_{n+1}} f(x, y(x))dx = hf(\xi, y(\xi)) \qquad (2)$$

Where $h = x_{n+1} - x_n$, $\xi \in [x_n, x_{n+1}]$, i.e.,

$$y(x_{n+1}) = y(n) + hf(\xi, y(\xi)).$$

If we approximate $f(\xi, y(\xi))$ by the linear combination values $f(\xi_1, y(\xi_1)), f(\xi_2, y(\xi_2)), \ldots, f(\xi_m, y(\xi_m))$ of $f(x, y(x))$ on the interval $[x_n, x_{n+1}]$, then arrive at the general form of Runge–Kutta method:

$$y_{n+1} = y_n + h\sum_{i=1}^{n} c_i f(\xi_i, y(\xi_i)) \quad (3)$$

By choosing different values of the parameters m, c_i and ξ_i we can get different forms of the Runge-Kutta computation formula. Just choose suitable values for the parameters and you can get a higher-order Runge-Kutta computation formula.

The fundamental premise of the Runge-Kutta method is to minimise the discrepancy between the estimated value of $y(t)$ and the actual result by making a series of intermediate estimates rather than relying on a single initial estimate. The Runge-Kutta method of order 4, the basis for ode45, employs these intermediate estimates, thereby enhancing the accuracy of the solution.

The ode45 function is a commonly used component of the MATLAB software and is employed primarily for the resolution of differential equations. It is based on the 4th and 5th order Runge-Kutta methods. This technique may be defined as a kind of numerical integration method used to solve differential equations and is suitable for initial value problems.

Definition 2.2 Dormand-Prince (4,5) Method and ode45 [27]

MATLAB's ode45 function employs the Dormand-Prince 4th and 5th order Runge-Kutta pair. In this method:
The 4th-order solution y_4 and the more accurate 5th-order solution y_5 are computed. Error control is performed using:

$$E = \|y_5 - y_4\| \quad (4)$$

where E is the estimated local error. If E exceeds a predefined tolerance threshold, the step size is reduced; otherwise, it is increased. This adaptive step-size strategy makes ode45 highly efficient for solving a wide range of differential equations. Dormand-Prince Method Coefficients The coefficients used by ode45 are summarized in the following table:

c_i	a_{ij}	$b_i^{(4)}$	$b_i^{(5)}$
0	-	0	0
1/5	1/5	1/5	1/5
3/10	3/40, 9/40	3/10	3/10
4/5	44/45, -56/15, 32/9	4/5	4/5
8/9	19372/6561, -25360/2187, 64448/6561, -212/729	8/9	8/9
1	9017/3168, -355/33, 46732/5247, 49/176, -5103/18656	1	1
1	35/384, 0, 500/1113, 125/192, -2187/6784, 11/84	35/384, 0, 500/1113, 125/192, -2187/6784, 11/84, 0	

Table 2.1. Dormand-Prince (4,5) Method Coefficients

Where $b_i^{(4)}$ and $b_i^{(5)}$ are the weight coefficients for the 4th and 5th-order solutions, respectively. ode45 compares these two solutions to determine the adaptive step size adjustment.

Theorem 2.3 (**Convergence of the 4th-Order Runge-Kutta Method)**
Consider the initial value problem (IVP)

$$y'(t) = f(t, y(t)), \qquad y(t_0) = y_0 \quad (5)$$

where the function $f : [t_0, T] \times \mathbb{R}^n \to \mathbb{R}^n$ is Lipschitz continuous in y and is at least four times continuously differentiable (i.e.,$f \in C^4$). Under these assumptions, the (5) IVP admits a unique, sufficiently differentiable solution y*(t)*.

Proof. The classical 4th-order Runge-Kutta method (RK4) for a step size h is defined by

$$k_1 = f(t_n, y_n)$$

$$k_2 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$$

$$k_3 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$$

$$k_4 = f(t_n + \frac{h}{2}, y_n + \frac{h}{2}k_3$$

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4). \quad (6)$$

Then, for sufficiently small *h* the global error of the RK4 method satisfies

$$\max_{0 \le n \le N} \|y(t_n) - y_n\| \le \widetilde{C}\, h^4\,, \quad (7)$$

where $\widetilde{C} > 0$ is a constant that depends continuously on the norms of f and its derivatives over the interval $[t_0, T]$. In other words, the method is globally 4th-order convergent.

Now, let us present the proof in a step-by-step detailed mannern.

Step1. For the exact solution $y(t)$, perform a Taylor series expansion about t_n :

$$y(t_{n+1}) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y^{(3)}(t) + \frac{h^4}{24}y^{(4)}(t) + \frac{h^5}{120}y^{(5)}(t)(\xi_n)\,, \quad (8)$$

where $\xi_n \in [t_n, t_{n+1}]$ and the derivatives $y^{(k)}$ exist due to the regularity of f. The RK4 update operator, denoted by

$$\Phi(t_n, y_n, h) = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4), \quad (9)$$

is constructed so that, when $y_n = y(t_n)$, the quantities k_i (which depend on f) can be expanded in Taylor series about t_n. By matching the coefficients in the Taylor expansion, it can be shown that

$$\Phi(t_n, y(t_n), h) = y(t_n) + hy'(t_n) + \frac{h^2}{2}y''(t_n) + \frac{h^3}{6}y^{(3)}(t) + \frac{h^4}{24}y^{(4)}(t) + \mathcal{O}(h^5)\,. \quad (10)$$

where, $\mathcal{O}(h^5)$ denotes the remainder term in the expansion, which satisfies $\mathcal{O}(h^5) \le \widetilde{C}\, h^5$ for some constant $\widetilde{C} > 0$ independent of h as $h \to 0$. Define the local truncation error τ_n as:

$$\tau_n = \frac{y(t_{n+1}) - \Phi(t_n, y(t_n), h)}{h}\,. \quad (11)$$

Substituting the Taylor expansion for $y(t_{n+1})$ and the expansion of $\Phi(t_n, y(t_n), h)$ we obtain:

$$\tau_n = \mathcal{O}(h^4). \quad (12)$$

This implies that the error made in one step (prior to division by h) is $\mathcal{O}(h^5)$.

Step 2. Let the global error at the n-th step be:

$$e_n = y(t_n) - y_n. \tag{13}$$

Our objective is to show that there exists a constant $\widetilde{C}$ such that

$$\max_{0\le n\le N} \|e_n\| \le \widetilde{C}\, h^4, \tag{14}$$

for all n such that $t_n \le T$. Starting from the exact evolution of the solution:

$$y(t_{n+1}) = \Phi(t_n, y(t_n), h) + h\tau_n, \tag{15}$$

and noting that the numerical solution is given by $y(t_{n+1}) = \Phi(t_n, y(t_n), h)$ we write the error at the next step as:

$$e_{n+1} = y(t_{n+1}) - y_{n+1} = \Phi(t_n, y(t_n), h) - \Phi(t_n, y_n, h) + h\tau_n. \tag{16}$$

Since f is Lipschitz continuous in y, the operator Φ inherits a local Lipschitz property:

$$\|\Phi(t_n, y(t_n), h) - \Phi(t_n, y_n, h)\| \le (1 + Lh)\|y(t_n) - y_n\| = (1 + Lh)\|e_n\|, \tag{17}$$

where L is the Lipschitz constant of f.

Step 3. Esablishing the Recursive Inequalityt we combine the previous results and we obtain:

$$\|e_{n+1}\| \le (1 + Lh)\|e_n\| + Ch^5, \tag{18}$$

where C is a constant stemming from the bound on τ_n. Applying the discrete version of Grönwall's inequality to the recursive inequality yields:

$$\|e_n\| \le \exp\big(L(t_n - t_0)\big)\,\|e_0\| + \frac{Ch^5}{Lh}\big[\exp\big(L(t_n - t_0)\big) - 1\big]. \tag{19}$$

Since the initial error $e_n = 0$, it follows that:

$$\|e_n\| \le \widetilde{C}\, h^4, \tag{20}$$

where the constant $\widetilde{C}$ depends on L, the final time T and the constant C. The error made in one RK4 step is $\mathcal{O}(h^5)$, which implies a normalized local error of $\mathcal{O}(h^4)$. Through the use of the Lipschitz condition and the discrete Grönwall inequality, the accumulation of local errors results in a global error that is bounded by $\widetilde{C}\, h^4$. Thus, even for differential equations without a closed-form solution, the classical 4th-order Runge-Kutta method converges with order 4 as the step size h tends to zero.

Theorem 2.4 Let us consider the second-order neutral functional differential equation with distributed delay:

$$\frac{d^2}{dt^2}\Big[y(t) - \int_0^t K(t,s)y(s-\tau)ds\Big] + p(t)\frac{dy}{dt} + q(t)y(t) = f(t), \tag{21}$$

with initial conditions:

$$y(t) = \phi(t), \qquad y'(t) = \psi(t) \qquad for\ t\epsilon\,[-\tau, 0], \tag{22}$$

Where $K(t,s), p(t), q(t)$ and $f(t)$ are continuous on $[0, \mathrm{T}]$ and $\phi, \psi\,\epsilon\, C([-\tau, 0])$. Then the solution $y_h(t)$ obtained by the fourth-order Runge-Kutta method combined with spline interpolation for the delay term and numerical quadrature for the integral satisfies:

i. $y_h(t) \to y(t)$ uniformly on $[0, \mathrm{T}]$ as $h \to 0$,

ii. $\|y(t) - y_h(t)\|_\infty = \mathcal{O}(h^4)$,

iii. The numerical scheme is stable with respect to small perturbations in the data.

Proof.

Step 1. We define

$$z(t) \coloneqq \frac{d}{dt}\left[y(t) - \int_0^t K(t,s)y(s-\tau)ds\right]. \tag{23}$$

Then we can transform equation (23) into a system:

$$\begin{cases} y'(t) = z(t) + \int_0^t \frac{\partial K}{\partial t}(t,s)y(s-\tau)ds - K(t,t)y(t-\tau), \\ z'(t) = f(t) - p(t)z(t) - q(t)y(t). \end{cases} \tag{24}$$

By classical theory [28], system (24) has a unique classical solution $y(t)\,\epsilon\, C^2([0,T])$.

Now we define h as the fixed step size, $t_n = nh$ and $y_h(t_n) \approx y(t_n)$. We use:

i) Fourth-order Runge-Kutta for the equivalent first-order system.

ii) Spline interpolation to estimate delayed values $y(t_n - \tau)$.

iii) Trapezoidal rule to approximate the integral $\int_0^{t_n} K(t_n,s)y(s-\tau)ds$.

Step 2. To illustrate the consistency of the method we get:

i. Runge-Kutta: Local truncation error is $\mathcal{O}(h^5)$, global error is $\mathcal{O}(h^4)$, under regularity of the solution.

ii. Spline interpolation error: Since $y(t)\,\epsilon\, C^2$, the error in approximating $y(s-\tau)$ by spline interpolation is bounded by:

$$|y(s-\tau) - \tilde{y}(s-\tau)| \leq C_1 h^2. \tag{25}$$

iii. Trapezoidal quadrature error: Since $K(t,s)$ and $y(s-\tau)$ are continuous:

$$\left|\int_0^t K(t,s)\, y(s-\tau)ds - \sum_j w_j K(t,s)\, \tilde{y}(s_j-\tau)\right| \leq C_2 h^2. \tag{26}$$

Thus, the overall local consistency error is of the order:

$$\mathcal{O}(h^2) + \mathcal{O}(h^2) + \mathcal{O}(h^5) = \mathcal{O}(h^2),$$

but the dominant contribution to the global solution error still comes from the Runge-Kutta method, so the global error remains $\mathcal{O}(h^4)$.

Step 3. Let us define the global error:

$$E_n \coloneqq |y(t_n) - y_h(t_n)|.$$

We use a discrete Grönwall inequality to analyze how the error grows with n.

Let:

$$E_{n+1} \leq (1 + Ch)E_n + Ch^5.$$

Then, iterating:

$$E_n \leq Ch^4(exp(CT) - 1),$$

which implies:

$$\|y(t) - y_h(t)\|_\infty = \mathcal{O}(h^4). \tag{27}$$

Hence, the method is numerically stable and convergent.

Step 4. In order to demonstrate the stability of the equation in relation to perturbations, it is first necessary to perturb the equation:

$$f(t) \to f(t) + \varepsilon(t), \quad \|\varepsilon\|_\infty \leq \varepsilon_0 .$$

Then, the solution $\tilde{y}$ satisfies:

$$|y(t) - \tilde{y}(t)| \leq C\varepsilon_0 ,$$

due to the boundedness of the coefficients and the linearity of the equation. Similarly, the numerical method will reflect this, with:

$$\|y(t) - \tilde{y}_h(t)\|_\infty \leq C'\varepsilon_0. \tag{28}$$

Consequently, the analysis confirms that the numerical scheme exhibits robustness and stability under small perturbations in the data or initial functions. Thus, the proof is rigorously completed.

3.Applied Examples

Example 3.1 Consider equation

$$y'' + ay^3 + bsin(ky) = 0 \tag{29}$$
$$y(0) = 1 \quad y'(0) = 0$$

initial value condition where a and b are positive fixed parameters. a, b and k parameters are critical factors determining the dynamics of the given nonlinear differential equation. The values of these parameters have a critical impact on the stability, oscillation frequency and overall dynamic behaviour of the system. (29) equation is analytically unsolvable because it contains nonlinear terms The nonlinear equations create complex dynamics and nonlinear equations often require solutions that are difficult to estimate; moreover, solution methods often require small variable assumptions and these assumptions may not be valid. Therefore, it is recommended to use numerical methods to solve such equations.

Let solve this equation using matlab to see the solutions of this equation in different parameters:

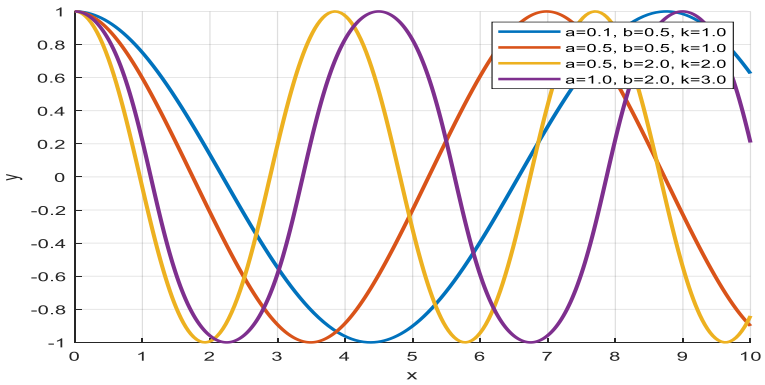

Figure 3.1 Solution of (29) Equation (According to Parameters)

The fluctuations seen in the graphs show the stability of the equilibrium points and the oscillations of the system; the change of y value over time reveals the dynamic evolution of the system and its tendency towards equilibrium.

Set No	a	b	k	Expected Behavior
1	0.1	0.5	1.0	Smooth oscillations, regular
2	0.5	0.5	1.0	Moderately damped oscillations
3	0.5	2.0	2.0	Strongly damped or chaotic transitions
4	1.0	2.0	3.0	Chaotic or irregular oscillations

Table 3.1 Parameter Sets and Expected Behavior

Example 3.2 Consider second-order nonlinear neutral differential equation of form

$$y'' + ay' + by + csin(y) + dy(x-\tau) = 0 \tag{30}$$

y'' : It usually represents acceleration or change in speed, ay' : The term damping may be used; this term can indicate friction or resistance in the system, by : It can act as a return force or spring force for the system (e.g. simple harmonic motion term), $csin(y)$: Adds a non-linear, periodic force or effect to the system. The sine term causes the system to deviate from linearity and can be seen, for example, in the motion of a pendulum in mechanics. $dy(x-\tau)$: $(x-\tau)$ is the value of y, τ time units earlier. This term means the system depends on both its current state and its state τ units ago. τ: delay duration, d: strength of the delayed effect. Because of this delay, future y values depend on past values, making the equation harder to solve.

Since (30) is non-linear, finding a general closed-form solution is difficult or impossible. To study the oscillatory behaviour of the given neutral differential equation, we can run simulations in MATLAB using different parameters and examine it in different solving ranges. The code below will solve the equation with different values of the parameters in order to observe the oscillatory or non-oscillatory behaviour of the system and to plot the results.

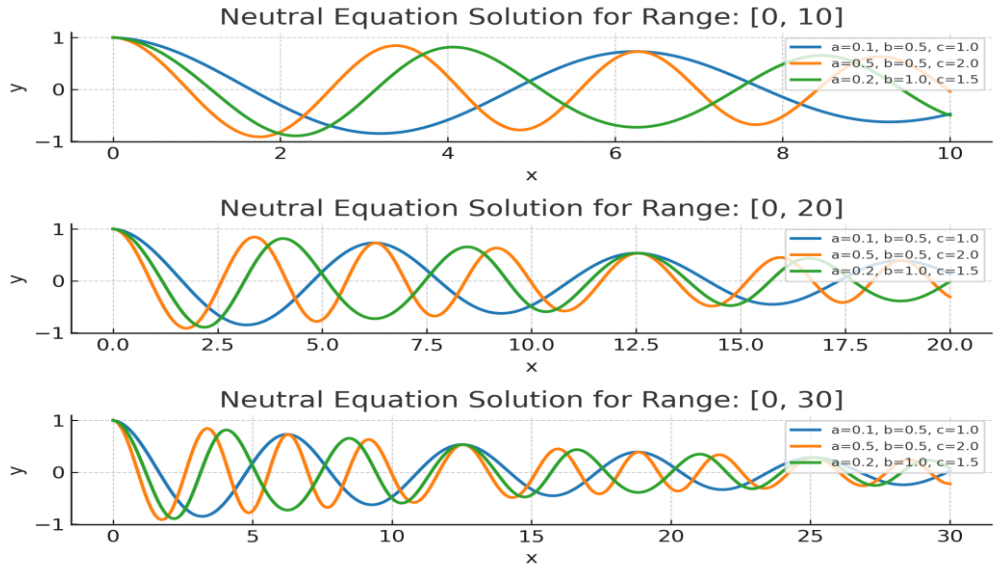

Figure 3.2 Behavior of (30) Equation for Different Parameter Sets and Ranges

The plotted solutions correspond to (30) equation solved numerically for various parameter sets (a, b, c) and over increasing domains.

The solutions exhibit sustained oscillatory behavior, which is characteristic of systems with combined nonlinear and delay effects. The inclusion of the nonlinear term $sin(y)$ contributes to the complexity and variability of the waveform, while the delayed feedback term $y(x-\tau)$ introduces a memory-dependent modulation that affects both the amplitude and phase of oscillations.

As the damping parameter a increases, a more rapid decay in amplitude is observed, indicating stronger energy dissipation in the system. Similarly, higher values of c amplify the nonlinearity, resulting in more pronounced deviations from harmonic behavior.

Overall, the solutions remain bounded, and the qualitative dynamics reflect the intricate interplay between damping, nonlinearity, and delay. These features are typical of neutral functional differential equations and highlight the rich dynamics that can arise even in relatively simple formulations.

Example 3.3 The following system presents a structure similar to a kind of population dynamics or ecological balance model

$$\frac{dx}{dt} = \alpha x\left(1-\frac{x}{K}\right) - \beta xy$$

$$\frac{dx}{dt} = \delta xy - \gamma y^2 \tag{31}$$

Where x, population of the first species (e.g. prey); y, population of the second species (e.g. predator); $\alpha, \beta, \delta, \gamma$ parameters are positive constants and K, Carrying capacity. (31) system of differential equations represents a population dynamic model, specifically a prey-predator model, wherein the dependent variables are the population sizes of the prey and the predator, respectively [29]. These types of models are employed to examine the interactions between prey and predator species in ecosystems. The analytical solution of the system (31) is very difficult due to the nonlinear terms, the complexity of the pairwise interactions between the populations and the large number of parameters. For this reason, a 3D matlab solution is given below. The ODE solution is indicated by a red line.

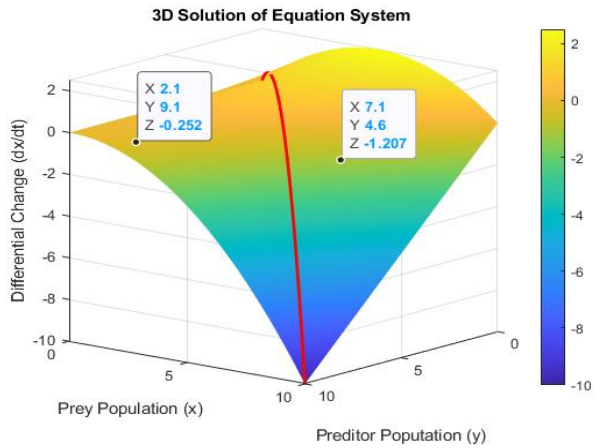

Figure 3.3a 3D Solution of (31) Equation System

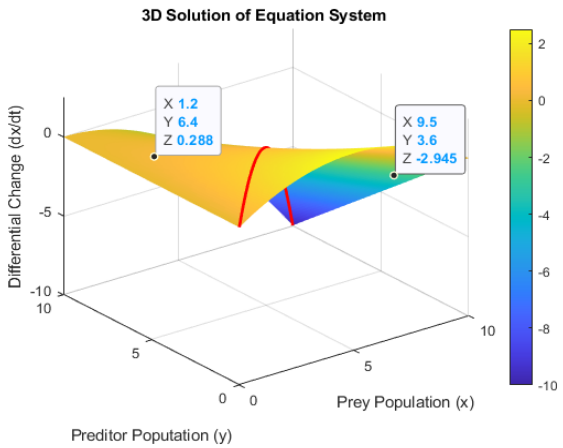

Figure 3.3b 3D Solution of (31)Equation System

Blue regions: $\frac{dx}{dt} < 0$, meaning the prey population is decreasing. Red regions: : $\frac{dx}{dt} < 0$, meaning the prey population is increasing. Middle regions (green/yellow): The prey population is approaching equilibrium. The prey $(x)$and predator (y) populations exhibit periodic oscillations. If the prey population increases significantly,

the predator population also rises. However, as predators consume more prey, the prey population eventually declines again. The solution of the system reveals bounded, oscillatory dynamics characteristic of predator–prey interactions. Population trajectories spiral around a nontrivial equilibrium, suggesting the presence of a limit cycle or center. These recurrent fluctuations arise from nonlinear feedback between species: increased prey supports predator growth, which in turn suppresses the prey, creating a natural cyclic rhythm. The inclusion of logistic growth (via the carrying capacity K) ensures bounded prey growth, while quadratic predator mortality prevents unbounded predator expansion. The overall system behavior reflects stability through dynamic equilibrium.

Example 3.4 Consider the third-order nonlinear differential equationof form

$$\frac{d^3x}{dt^3} + \alpha\frac{d^2x}{dt^2} + \beta\frac{dx}{dt} + \gamma x + \delta\sin(x) + \sigma\cos(x) + \mu\sin(\omega x) = 0 \tag{32}$$

$\alpha\frac{d^2x}{dt^2}$: This term depends on the system's velocity (first derivative) and acceleration (second derivative). The acceleration $\frac{d^2x}{dt^2}$ represents the changing motion over time and models the damping effect in the system, i.e., the movement of the system decreases over time.

$\beta\frac{dx}{dt}$: This term represents a force that is proportional to the velocity of the system. As the velocity $\frac{dx}{dt}$ increases, the resistance encountered by the system increases, slowing it down. This also contributes to the damping behavior of the system.

γx: This term represents the feedback effect of the system. A positive value of γ\gammaγ creates a restoring force proportional to the displacement x. The larger the displacement, the larger the force acting to return the system to equilibrium.

$\delta\sin(x)$: This term is a nonlinear force that depends on the position of the system. As xxx increases, this term becomes more complex and introduces nonlinear behavior into the system. These types of terms are often seen in oscillator systems and cause the system to exhibit more complex oscillations.

$\sigma\cos(x)$: This term also represents a nonlinear force, but this time, it depends on the cosine of the position. The presence of these nonlinear terms leads to more complex behavior, such as oscillations, in the system's solution.

$\mu\sin(\omega x)$: This term represents an external forcing term that varies with time. The parameter μ controls the amplitude of the external force, and ω controls its frequency. This external forcing influences the system's motion, adding external oscillations on top of the system's natural dynamics.

The nonlinear terms in the solution contribute to the system's instability and complexity. Additionally, the time-varying forcing terms (sine function) account for external influences and environmental factors, altering the system's behavior. Such systems are often referred to as dynamic systems, oscillators, or systems under forcing effects. Instead of an analytical solution, these types of equations are generally studied using numerical solutions (such as ODE solvers). The lack of an analytical solution for this equation arises from both the nonlinear dynamics and the complex external forcing effects. The system is too complex, and numerical solutions are required to obtain the correct solution. Such systems are commonly found in many real-world physical models, particularly in complex dynamics observed in engineering and natural sciences.

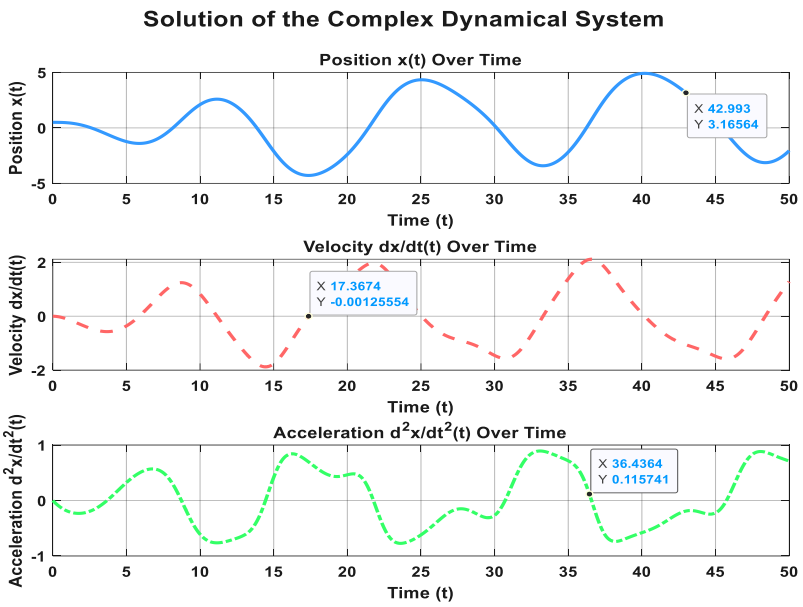

Figure 3.4 Solutions of (32) for acceleration, velocity and position.

Solving this equation is crucial for understanding how the system behaves over time. Such an equation typically exhibits complex oscillations and periodic behaviors. The nonlinear terms and external forcing make the system's behavior difficult to solve analytically. Therefore, numerical methods (such as ODE solvers) are typically used to study these types of equations.

Velocity and Acceleration: The graph also shows how velocity $\frac{dx}{dt}$ and acceleration $\frac{d^2x}{dt^2}$ change over time. This allows us to understand not only the position of the system but also the dynamics of its motion. For example, when acceleration is positive, the system speeds up, and when acceleration is negative, it slows down.

External Forcing: The forcing term $\mu \sin(\omega x)$ adds external energy to the system. This external forcing interacts with the system's natural oscillations and leads to more complex behaviors in the solution.

Conclusion

In this paper, we have used MATLAB to search for solutions to some differential equations that simply cannot be expressed in a solvable form or whose solutions are quite complex and visualised the solutions of these equations. The results obtained emphasise the importance of numerical methods for understanding and analysing the dynamics of complex systems. The originality of our work is that it both focuses on equations that have not been previously studied in certain areas and makes complex dynamics more understandable by visualising these equations. Furthermore, detailing the methodologies used in solving such systems provides a basis for future research. Thus, it allows for more in-depth studies on topics such as complex population dynamics or the behaviour of physical systems.

Acknowledgment

The author extends gratitude to the referees for their thorough review, which significantly enhanced the manuscript and to the readers for their interest and engagement with the work.

Matlab code for Example 3.1	Matlab code for Example 3.2	Matlab code for Example 3.3	Matlab code for Example 3.4

```
paramSets = [
   0.1, 0.5, 1.0;  % Set 1: a = 0.1, b = 0.5, k = 1.0
   0.5, 0.5, 1.0;  % Set 2: a = 0.5, b = 0.5, k = 1.0
   0.5, 2.0, 2.0;  % Set 3: a = 0.5, b = 2.0, k = 2.0
   1.0, 2.0, 3.0   % Set 4: a = 1.0, b = 2.0, k = 3.0
];
% initial conditions
y0 = 1;    % y(0) initial value
dy0 = 0;   % dy/dx(0) initial value
% solution interval
xspan = [0 10];
% ODE settings
options = odeset('RelTol',1e-6,'AbsTol',1e-8);
figure;
hold on;

for i = 1:size(paramSets, 1)
   a = paramSets(i, 1); % a parameter
   b = paramSets(i, 2); % b parameter
   k = paramSets(i, 3); % k parameter

  odefun = @(x, Y) [Y(2); -a * Y(1)^3 - b * sin(k * Y(1))];

   try
      [T, Y] = ode45(odefun, xspan, [y0; dy0], options);

     if size(Y, 2) < 2
        error('The dimensions of the solution matrix are lower than expected.');
      end

      plot(T, Y(:, 1), 'LineWidth', 2, 'DisplayName', sprintf('a=%.1f, b=%.1f, k=%.1f', a, b, k));

   catch ME
      fprintf('error: %s\n', ME.message);
   end
end

xlabel('x');
ylabel('y');
legend('show');
grid on;
hold off;
```

```
function solve_neutral_eq()

   paramSets = [

      0.1, 0.5, 1.0;  % Set 1: a = 0.1, b = 0.5, c = 1.0

      0.5, 0.5, 2.0;  % Set 2: a = 0.5, b = 0.5, c = 2.0

      0.2, 1.0, 1.5;  % Set 3: a = 0.2, b = 1.0, c = 1.5

   ];

  tau = 0.5;

   d = 0.3; % Delay coefficient

  % Initial conditions

   y0 = 0.5;

   dy0 = 0.1;

  xspan_set = [

      0, 10;

      0, 20;

      0, 30

   ];

  options = odeset('RelTol',1e-6,'AbsTol',1e-8);

figure;

   for j = 1:size(xspan_set, 1)

      subplot(3, 1, j);

      hold on;

  for i = 1:size(paramSets, 1)

         a = paramSets(i, 1);

         b = paramSets(i, 2);

         c = paramSets(i, 3);

    % Initial history storage

         T_hist = [];  % Empty list for time points

         Y_hist = [];  % Empty list for solution values

   % Solve the equation

         [T, Y] = ode45(@(x, Y) odefun_with_history(x, Y, a, b, c, d, tau, T_hist, Y_hist, y0), xspan_set(j, :), [y0; dy0], options);

         T_hist = T;

         Y_hist = Y(:,1);

     plot(T, Y(:, 1), 'LineWidth', 2);

      end

   xlabel('x');

      ylabel('y');

      title(['Neutral Equation Solution for Range: [', num2str(xspan_set(j, 1)), ', ', num2str(xspan_set(j, 2)), ']']);

       legend({'a=0.1, b=0.5, c=1.0', 'a=0.5, b=0.5, c=2.0',
```

```
alpha = 1;    % Growth rate of prey

beta = 0.1;  % The effect of the predator on the prey

gamma = 1;     % Predator's natural mortality

delta = 0.1; % Predator's growth rate from prey

K = 10;     % Carrying capacity


tspan = [0 50];

y0 = [5; 2];    % initial conditions


% ODE solution

[t, y] = ode45(@(t, y) predatorPreySystem(t, y, alpha, beta, gamma, delta, K), tspan, y0);


 [x, y] = meshgrid(0:0.1:10, 0:0.1:10);

z = alpha*x.*(1 - x/K) - beta*x.*y; % Population dynamics surface


figure;

surf(x, y, z, 'EdgeColor', 'none');

colorbar;


xlabel('Prey Population (x)');

ylabel('Preditor Poputation (y)');

zlabel('Differential Change (dx/dt)');

title('3D Solution of Equation System ');

view(3);

axis tight;


hold on;

plot3(y(:,1), y(:,2), alpha*y(:,1).*(1 - y(:,1)/K) - beta*y(:,1).*y(:,2), 'r', 'LineWidth', 2);

hold off;


function dydt = predatorPreySystem(t, y, alpha, beta, gamma, delta, K)

   x = y(1);

   y_val = y(2);


   dxdt = alpha*x*(1 - x/K) - beta*x*y_val;
```

```
% Parameters

alpha = 0.5;  % Coefficient for the second derivative term

beta = 0.2;  % Coefficient for the first derivative term

gamma = 0.1;  % Positive linear term

delta = 0.3;  % Coefficient for the sine term

epsilon = 0.1; % Coefficient for the cosine term

eta = 0.05;  % Coefficient for the external forcing term

omega = 1;   % Frequency of the external force

% Initial conditions

y0 = [0.5; 0; 0];  % x(0) = 0.5, dx/dt(0) = 0, d^2x/dt^2(0) = 0

% Time span

tspan = [0 50];

t_eval = linspace(tspan(1), tspan(2), 1000);

% ODE solver

odefun = @(t, Y) [Y(2);          % dx/dt

          Y(3);          % d^2x/dt^2

          -alpha*Y(3) - beta*Y(2) - gamma*Y(1) - delta*sin(Y(1)) - epsilon*cos(Y(1)) + eta*sin(omega*t)];          % d^3x/dt^3

 [t, Y] = ode45(odefun, t_eval, y0);


% Create figures

figure('Position', [100, 100, 900, 700], 'Color', 'w');


subplot(3, 1, 1);

plot(t, Y(:,1), 'LineWidth', 2.5, 'Color', [0.2 0.6 1], 'LineStyle', '-');

xlabel('Time (t)', 'FontSize', 14, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);

ylabel('Position x(t)', 'FontSize', 14, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);

title('Position x(t) Over Time', 'FontSize', 16, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);

grid on;

set(gca, 'FontSize', 12, 'FontWeight', 'bold', 'GridAlpha', 0.3);
```

	'a=0.2, b=1.0, c=1.5'}, 'Location', 'Best');
	grid on;
	hold off;
	end
	end
	%% Delayed Differential Equation Function
	function dYdx = odefun_with_history(x, Y, a, b, c, d, tau, T_hist, Y_hist, y0)
	if x <= tau
	y_tau = y0; % In the initial region, y(tau) = y(0)
	else
	if isempty(T_hist)
	y_tau = y0;
	else
	y_tau = interp1(T_hist, Y_hist, x - tau, 'linear', 'extrap');
	end
	end
	% Define the differential equation
	dYdx = [Y(2); -a * Y(2) - b * Y(1) - c * sin(Y(1)) - d * y_tau];
	end
		dydt1 = delta*x*y_val - gamma*y_val^2;
		dydt = [dxdt; dydt1];
		end
			set(gca, 'XColor', [0.1 0.1 0.1], 'YColor', [0.1 0.1 0.1]);
			subplot(3, 1, 2);
			plot(t, Y(:,2), 'LineWidth', 2.5, 'Color', [1 0.4 0.4], 'LineStyle', '--');
			xlabel('Time (t)', 'FontSize', 14, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);
			ylabel('Velocity dx/dt(t)', 'FontSize', 14, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);
			title('Velocity dx/dt(t) Over Time', 'FontSize', 16, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);
			grid on;
			set(gca, 'FontSize', 12, 'FontWeight', 'bold', 'GridAlpha', 0.3);
			set(gca, 'XColor', [0.1 0.1 0.1], 'YColor', [0.1 0.1 0.1]);
			subplot(3, 1, 3);
			plot(t, Y(:,3), 'LineWidth', 2.5, 'Color', [0.2 1 0.4], 'LineStyle', '-.');
			xlabel('Time (t)', 'FontSize', 14, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);
			ylabel('Acceleration d^2x/dt^2(t)', 'FontSize', 14, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);
			title('Acceleration d^2x/dt^2(t) Over Time', 'FontSize', 16, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);
			grid on;
			set(gca, 'FontSize', 12, 'FontWeight', 'bold', 'GridAlpha', 0.3);
			set(gca, 'XColor', [0.1 0.1 0.1], 'YColor', [0.1 0.1 0.1]);
			sgtitle('Solution of the Complex Dynamical System', 'FontSize', 18, 'FontWeight', 'bold', 'Color', [0.1 0.1 0.1]);
			set(gcf, 'Color', 'w');

Table 3.1 Matlab Codes for Examples

References

[1] Yang, W. Y., Cao, W., Kim, J., Park, K. W., Park, H. H., Joung, J., ... & Im, T. 2020. Applied numerical methods using MATLAB, John Wiley & Sons.

[2] Corless, R. M., Nicolas, F. 2013. A graduate introduction to numerical methods, *AMC* (2013) 10: 12.

[3] Denis, B. 2020. An overview of numerical and analytical methods for solving ordinary differential equations, arXiv preprint arXiv*:* 2012.07558.

[4] Yüzbasi, S., Gök, E., Sezer, M. 2016. A numerical method for solving systems of higher order linear functional differential equations, Open Physics, 14(1) (2016) 15-25.
[5] Saqib, M., et al. 2024. Dynamical Behavior of Nonlinear Coupled Reaction-Diffusion Model: A Numerical Study Utilizing ADI and Staggered Grid Finite Volume Method in Matlab, IEEE Access.
[6] Gopal, D., et al. 2021. Numerical analysis of higher order chemical reaction on electrically MHD nanofluid under influence of viscous dissipation, Alexandria Engineering Journal, 60.1 (2021) 1861-1871.
[7] Shah, K., Fahd, J., Thabet, A. 2020. Stable numerical results to a class of time-space fractional partial differential equations via spectral method, Journal of Advanced Research, 25 (2020) 39-48.
[8] Ong, B. W., Spiteri, R. J. 2020. Deferred correction methods for ordinary differential equations, Journal of Scientific Computing, *83*(3) (2020) 60.
[9] Christlieb, A., Ong, B., Qiu, J.M. 2009. Comments on high-order integrators embedded within integral deferred correction methods, Commun. Appl. Math. Comput. Sci., 4 (2009) 27–56.
[10] Christlieb, A., Ong, B., Qiu, J.M., Integral deferred correction methods constructed with high order Runge–Kutta integrators, *Math. Comput.*, 79(270) (2010) 761–783.
[11] Christlieb, A.J., Macdonald, C.B., Ong, B.W. 2010. Parallel high-order integrators, SIAM J. Sci. Comput., 32(2) (2010) 818–835.
[12] Dutt, A., Greengard, L., Rokhlin, V. 2000. Spectral deferred correction methods for ordinary differential equations, *BIT Numerical Mathematics*, 40(2) (2000) 241–266.
[13] Fox, L., Goodwin, E.T. 1949. Some new methods for the numerical integration of ordinary differential equations, Proc. Camb. Philos. Soc., 45 (1949) 373–388.
[14] Hansen, A.C., Strain, J. 2011. On the order of deferred correction, *Appl. Numer. Math.,* 61(8) (2011) 961–973.
[15] Fox, L. 1947. Some improvements in the use of relaxation methods for the solution of ordinary and partial differential equations. Proc. R. Soc. Lond. Ser. A, 190 (1947) 31–59.
[16] Pereyra, V. 1966. On improving an approximate solution of a functionalequation by deferred corrections, Numerische Mathematik, 8 (1966) 376-391.
[17] Pereyra, V. 1968. Iterated deferred corrections for nonlinear boundary value problems, Numerische Mathematik, 11 (1968) 111– 125.
[18] Kress, W., Gustafsson, B. 2002. Deferred correction methods for initial boundary value problems, Proceedings of the 5th International Conference on Spectral and High Order Methods (ICOSAHOM-01) (Uppsala), vol. 17 (2002) 241–251.
[19] Rangan, A.V. 2003. Adaptive solvers for partial differential and differential-algebraic equations, Ph.D. thesis, University of California, Berkeley.
[20] Huang, J., Jia, J., Minion, M. 2006. Accelerating the convergence of spectral deferred correction methods, J. Comput. Phys., 214(2) (2006) 633–656.
[21] Chu, K.W., Spence, A. 1981. Deferred correction for the integral equation eigenvalue problem, *The ANZIAM Journal,* 22(4) (1981) 474–487.
[22] Reuter, B., et al. 2021. FESTUNG 1.0: Overview, usage, and example applications of the MATLAB/GNU Octave toolbox for discontinuous Galerkin methods, Computers & Mathematics with Applications, 81 (2021) 3-41.
[23] Shampine, L. F. 2018. Numerical solution of ordinary differential equations, Routledge.
[24] Cooper, J. M. 2012. Introduction to partial differential equations with MATLAB. Springer Science & Business Media.
[25] Coleman, M. P., Bukshtynov, V. 2024 An introduction to partial differential equations with MATLAB, CRC Press.
[26] Zheng, L., Zhang, X. 2017. Modeling and analysis of modern fluid problems. Academic Press.
[27] Dormand, J. R., & Prince, P. J. 1980. A family of embedded Runge-Kutta formulae, Journal of Computational and Applied Mathematics, 6(1), 19–26.
[28] Hale, J. K. 2006. Functional differential equations. In: Analytic Theory of Differential Equations: The Proceedings of the Conference at Western Michigan University, Kalamazoo, from 30 April to 2 May 1970. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006. p. 9-22.
[29] Murray, J. D. 2007. Mathematical biology: I. An introduction (Vol. 17), Springer Science & Business Media, (2007).

[30] Zaya, N. E., Hassan, L. H., & Bilgil, H. 2018. Mathematical Modeling for Prediction of Heating and Air-Conditioning Energies of Multistory Buildings in Duhok City, Acad. J. Nawroz Univ, *7*, 153-167.
[31] https://doi.org/10.38016/jista.1447980