Dijlah Journal of Engineering Science (DJES) Vol. 2, No. 4, December, 2025, pp. 23-33
ISSN: Printed: **3078-9656, Online: 3078-9664**, paper ID: 76

Advanced Numerical Methods for Solving Nonlinear Differential Equations: Theory, Algorithms, and Applications

Nuha Nazal Mohammed
nuhanazalm@gmail.com

Article Info

Article history:

Received Aug. 1, 2025
Revised Sept., 20, 2025
Accepted Oct., 20, 2025

Keywords:

Numerical Methods, Nonlinear Differential Equations, Applications

ABSTRACT

Nonlinear differential equations (NDEs) are considered a crucial part in modeling complex physical, biological, and engineering systems. Classical numerical methods exhibit limitations in stability, accuracy, and efficiency. The purpose of this paper is to present a comprehensive review and analysis of advanced methods for solving NDEs. In this paper, we focused on recent numerical novel algorithms and laid the theoretical foundations and practical performance considerations.

Corresponding Author:

Nuha Nazal Mohammed
Email: nuhanazalm@gmail.com

1. INTRODUCTION

Nonlinear differential equations (NDEs) play a significant role in modeling complex physical, biological, and engineering systems. Examples include nonlinear oscillations in mechanical systems (Nayfeh and Mook), chaotic dynamics in atmospheric models (Lorenz), reaction–diffusion processes in chemical kinetics (Epstein and Pojman) , and population dynamics in ecology (Murray) . Analytical solutions to these equations are rare and often limited to special cases. Therefore, numerical methods have risen as an important substitute to find accurate solutions within reasonable constraints in time and memory. Traditional approaches for solving NDEs numerically include explicit and implicit Runge–Kutta methods (Butcher, Numerical Methods for Ordinary Differential Equations), linear multistep methods (Hairer and Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems), and spectral methods (Boyd) . While these methods are well-established and widely used, they have their own limitations when applied to strongly linear, stiff, or multi-scale systems. For instance, explicit schemes may require small time steps for stability, whereas implicit methods demand the repeated solution of nonlinear algebraic systems, which can be computationally expensive. Recent advances in numerical analysis and scientific computing have motivated the development of novel algorithms that aim to improve stability, accuracy, and efficiency for challenging nonlinear problems. Examples include adaptive time-stepping strategies based on local error estimation (Söderlind), exponential integrators for oscillatory and stiff systems (Hochbruck and Ostermann), operator splitting methods for multi-physics problems (Hundsdorfer and Verwer) , and structure-preserving schemes that conserve invariants of the underlying system (Hairer, Lubich and Wanner, Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations) . Additionally, the integration of machine learning techniques into numerical solvers has opened new possibilities for data-driven discretizations and hybrid analytical–numerical methods (Raissi, Perdikaris and Karniadakis) .

The purpose of this paper is to present a comprehensive review and analysis of advanced numerical methods for solving nonlinear differential equations. We begin with a discussion of mathematical preliminaries essential for understanding the properties of numerical schemes. We then examine the limitations of traditional methods and introduce modern approaches designed to overcome these challenges.

The contributions of this work are as follows:

1. A structured classification of advanced numerical algorithms for NDEs, highlighting their theoretical basis.
2. Comparative analysis of stability, convergence, and computational efficiency across different classes of methods.
3. Case studies demonstrating the performance of selected algorithms on representative nonlinear problems.

The remainder of the paper is organized as follows: Section 2 reviews the mathematical foundations relevant to numerical methods for solving differential equations. Section 3 discusses classical numerical methods in the literature for solving NDEs. Section 4 presents several advanced algorithms and analyzes their theoretical properties. Followed by a conclusion that recaps the whole paper.

2. METHODS AND MATHEMATICAL BACKGROUND

This section recaps the mathematical background needed in the remainder of the paper. We start by giving a definition of Non-linear differential equations (NDE), then discuss the stability, convergence, stiffness, and error norms of numerical analysis. These terms will be important for the coming sections.

a. Definition of non-linear differential equations

A *nonlinear differential equation* is a differential equation in which the dependent variable or its derivatives appear in a nonlinear manner. Let $u(x)$ denote the unknown function, and $u', u'', \dots, u^{(n)}$ its derivatives with respect to the independent variable x. The general form of an n-th order differential equation can be expressed as

$$F\left(x, u, u', u'', \dots, u^{(n)}\right) = 0$$

where F is a given function of its arguments.

The equation is said to be *linear* if it can be written as

$$a_n(x)\frac{d^n u}{dx^n} + a_{n-1}(x)\frac{d^{n-1}u}{dx^{n-1}} + \cdots + a_0(x)u = f(x)$$

where the coefficients $a_i(x)$ depend only on x and u along with its derivatives appear only to the first power, without being multiplied together. A *nonlinear* differential equation violates at least one of these conditions.

Nonlinear differential equations (NDE) can be classified according to several criteria, including the number of independent variables, the order of derivatives involved, and the nature of the nonlinearity.

Number of Independent Variables

- **Nonlinear Ordinary Differential Equations (ODEs):** Which involves derivatives of single independent variable. For example:

$$u'' + u^2 \ = 0 \, u'' - \mu(1-u^2)u' + u \ = 0$$

- **Nonlinear Partial Differential Equations (PDEs):** Involve partial derivatives with respect to two or more independent variables:

$$u_t + u\,u_x \ = 0$$

The order of the Equation

- **First-Order Nonlinear Differential Equations:** Equations containing only first derivatives but in nonlinear form:

$$u' = u^2 + sin(u)\, u_t + u\, u_x = 0$$

- **Higher-Order Nonlinear Differential Equations:** which contain second- or higher-order derivatives with nonlinear terms.

$$u'' + u^3 = 0, u_{tt} - \alpha u_{xx} + \beta u^3 = 0.$$

The nature of the nonlinearity

- **Polynomial Nonlinearity:** Nonlinear terms involve polynomial powers of the dependent variable or its derivatives.

$$u'' + u^2 = 0$$

- **Trigonometric or Exponential Nonlinearity:** Nonlinear terms involve transcendental functions.

$$u'' + sin(u) = 0, \quad u' = e^u$$

- **Multiplicative Nonlinearity:** Dependent variable and its derivatives appear as products.

$$u\, u' + x = 0, \quad (u \cdot \nabla)u$$

b. Principle of Numerical Solutions for Nonlinear Differential Equations

We can't find the exact solution of NDEs with a closed form, especially if they contain complex boundary and initial conditions. Instead, we use numerical methods that can approximate the solution iteratively by discretizing the continuous problem into a discrete one, which can be solved computationally.

An n-th order NDE can be expressed as follows:

$$F\left(t, u, u', u'', \dots, u^{(n)}\right) = 0,$$

where u is the unknown function, t represents one or more independent variables, and F is nonlinear in at least one of its arguments. The goal is to compute an approximation $\tilde{u}$ such that:

$$\tilde{u}(t) \approx u(t), \quad \forall t \in [t_0, T]$$

Instead of solving the continuous problem, the domain is replaced by a set of points:

$$t_0,\ t_1,\ t_2,\ \dots,\ t_N, \quad with \quad t_{n+1} = t_n + \Delta t$$

For Partial Differential Equations (PDEs), we discretize both the time and space domains, resulting in a computational grid.

We also need to discretize the derivatives to solve the equation numerically. Instead of analytically calculating the derivative, we replace it with a finite difference, finite element, or spectral approximation. For example, a first derivative can be formulated as :

$$u'(t_n) \approx \frac{u^{n+1} - u^n}{\Delta t} \quad (forward\ difference),$$

or, for higher accuracy:

$$u'(t_n) \approx \frac{u^{n+1} - u^{n-1}}{2\Delta t} \quad (central\ difference).$$

Nonlinear terms in F are retained in the discrete equations, leading to either:

- **Explicit schemes:** Compute u^{n+1} directly from known quantities at earlier steps.

- **Implicit schemes:** Require solving a nonlinear algebraic system at each time step, typically using iterative solvers such as Newton–Raphson or fixed-point iteration.

The choice of time step Δt and spatial discretization Δx must satisfy stability criteria, such as the Courant–Friedrichs–Lewy (CFL) condition for certain PDEs. The scheme should also be consistent and convergent, ensuring that:

$$\lim_{\Delta t,\Delta x\to 0} \tilde{u}(t) = u(t)$$

For nonlinear problems, convergence of the iterative solver at each step is essential. Adaptive time-stepping and mesh refinement may be employed to improve accuracy and efficiency.

c. Stability criteria of numerical algorithms

To understand the stability of numerical methods, let's take as an example the Initial Value Problem (IVP):

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0,$$

If f is smooth enough. If small adjustments to the starting data or intermediate calculations result in only small changes in the numerical answer, the numerical approach is considered *stable* (Hairer and Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems) .

d. Convergence criteria

A numerical method is *convergent* if the numerical solution y_n approaches a specific (fixed) value as the time step $h \to 0$. Also, the numerical method is considered "consistent" if this value is close to $y(t_n)$ for every t_n in the integration period (Thompson and Thompson). If τ_n refers to the *local truncation error* (LTE), then for a pth-order method we have

$$\tau_n = O(h^{p+1}),$$

and the *global error* satisfies

$$\| y(t_n) - y_n \| = O(h^p).$$

The *Lax–Richtmyer equivalence theorem* states that for a consistent method, stability is equivalent to convergence.

e. Stiffness criteria

An important characteristic that appears in many nonlinear problems, especially those involving widely separated time scales, is stiffness. Even when the solution evolves slowly over time, a system is said to be stiff if explicit numerical methods need incredibly short time steps to maintain stability. For the linear system:

$$\frac{dy}{dt} = Ay,$$

stiffness can also be described by the stiffness ratio

$$S = \frac{\max_i |R(\lambda_i)|}{\min_i |R(\lambda_i)|},$$

where λ_i are the eigenvalues of A. Large values of S indicate the existence of stiffness.

f. Norms and Error Measures criteria

Let y be the exact solution and y_h the numerical approximation on some mesh $\{t_n\}_{n=0}^{N}$. Common norms used to measure error include:

$$\| e \|_\infty = \max_{0\le n\le N} |y(t_n) - y_n|$$

$$\| e \|_2 = \left(\sum_{n=0}^{N} \quad |y(t_n) - y_n|^2 \right)^{1/2}$$

$$\| e \|_1 = \sum_{n=0}^{N} \quad |y(t_n) - y_n|$$

3. CLASSICAL NUMERICAL METHODS FOR SOLVING NONLINEAR DIFFERENTIAL EQUATIONS

Classical numerical methods for NDEs are inspired by their linear counterparts, with some added modifications to deal with nonlinear terms. In this section, we will discuss classical methods for solving NDEs of different types.

a. Nonlinear Ordinary Differential Equations (N-ODEs)

As an example of N-ODEs, consider the initial value problems (IVPs) of the form:

$$\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0$$

where $f(.)$ is a non linear function. Classical approaches for solving this problem include:

Euler’s Method (explicit and implicit)

Explicit Euler is straightforward but conditionally stable; implicit Euler requires solving nonlinear algebraic equations at each step (Hairer, Nørsett and Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems).

Runge–Kutta (RK) Methods

Higher-order methods like RK4 offer improved accuracy; implicit variants such as the implicit midpoint method enhance stability for stiff problems (Butcher, Numerical Methods for Ordinary Differential Equations).

Linear Multistep Methods

Explicit Adams–Bashforth and implicit Adams–Moulton schemes are widely used; *Backward Differentiation Formulas* (BDF) are effective for stiff systems (Hairer and Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems).

Predictor–Corrector Methods

Combine an explicit prediction with an implicit correction, balancing efficiency and stability (Hairer, Nørsett and Wanner, Solving Ordinary Differential Equations I: Nonstiff Problems).

In general, nonlinear implicit schemes require iterative solvers, such as Newton-Raphson or fixed-point iteration, at each step.

b. Nonlinear Partial Differential Equations (N-PDEs)

For N-PDEs, the *method of lines* discretizes space first, producing a system of ODEs. Classical spatial discretization strategies include:

Finite Difference Methods (FDM)

Approximate derivatives using finite difference quotients on structured grids (LeVeque).

Finite Element Methods (FEM)

Employ basis functions over mesh elements, suitable for complex geometries (Morton and Mayers) .

Spectral Methods

Use global basis functions like Fourier or Chebyshev polynomials for high accuracy in smooth problems (Trefethen, Spectral Methods in MATLAB) .

Time integration of the resulting ODEs can use any of the methods listed in the ODE section. For stiff nonlinear PDEs, implicit or semi-implicit methods (IMEX) are favored (Hairer and Wanner, Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems) .

c. Classification by Time-Stepping

From a computational and stability perspective:

Explicit Methods

Simple to implement but limited by stability restrictions, such as the Courant–Friedrichs–Lewy condition (LeVeque).

Implicit Methods

Often unconditionally stable but require solving nonlinear systems.

Semi-Implicit (IMEX) Methods

Handle stiff terms implicitly and non-stiff terms explicitly, offering a balance between stability and cost.

4. ADVANCED NUMERICAL METHODS FOR NONLINEAR PROBLEMS

After we reviewed relevant mathematical background about NDEs, then mentioned classical numerical methods for solving them, we will now discuss recent methods that are being used to solve NDEs in various problems. This section includes three main recent algorithms, namely: Extreme Learning Machines, Physics-Informed Neural Networks, and Fourier Neural Operators. All of these methods rely on the development of deep learning models, which are part of machine learning models that are used nowadays in solving hundreds of types of problems from different domains. We will discuss each method, and describe how it works and what its advantages are compared to classical methods.

a. Extreme Learning Machines (ELM)

With the advancement of machine learning, it has been used in many fields of science, including mathematics and differential equations. Extreme Learning Machines (ELM) are similar in structure to a Single Hidden Layer Feedforward Neural Network (SLFN), but instead of applying stochastic gradient descent, ELMs use matrix inversion to update the weights and biases of the network.

The general form of a nonlinear PDE can be written as

$$N\big(u(x)\big) = f(x), \quad x \in \Omega,$$

where N is a nonlinear differential operator, u is the unknown solution, f is a source term, and Ω denotes the problem domain. Appropriate boundary conditions are imposed on $\partial\Omega$.

The ELM framework approximates the solution $u(x)$ by a single-layer feedforward neural network of the form

$$u_N(x) = \sum_{i=1}^{N_h} \beta_i \, \sigma(w_i \cdot x + b_i)$$

where N_h is the number of hidden nodes, σ is an activation function, w_i and b_i are the randomly generated weights and biases of the hidden layer, and β_i are the trainable output weights determined by solving a linear system, and x is the input vector. The training algorithm goes as follows:

1. randomly assign weights w_i and bias b_i
2. calculate hidden layer output
3. calculate the output weights β_i
4. use β_i to make a prediction on new data x_j.

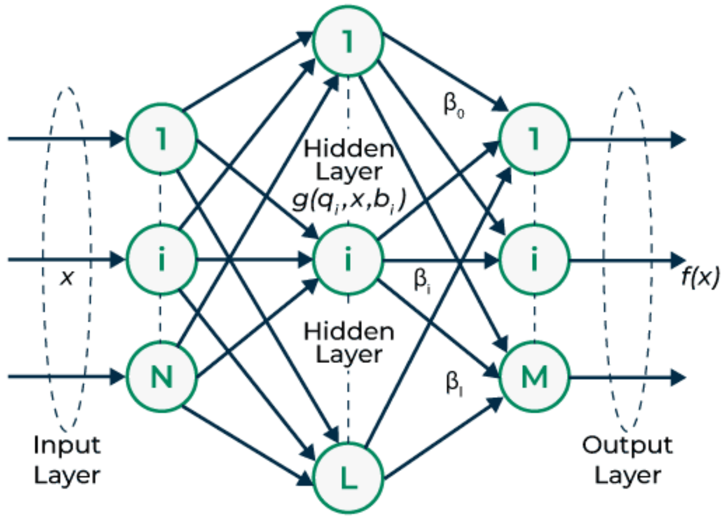

Figure 1 The overall architecture of Extreme Learning Machines.

The accuracy of the approximation is ensured by enforcing that the residual of the PDE vanishes at a set of collocation points $\{x_j\}_{j=1}^{M}$:

$$R(x_j) = N\left(u_N(x_j)\right) - f(x_j) \approx 0, \quad j = 1, \dots, M.$$

This results in a nonlinear system, which is solved using Newton-Raphson iterations. To analyze bifurcation structures, the authors in (Fabiani, Calabrò and Russo) employ a pseudo-arc-length continuation scheme:

$$F(u, \lambda) = 0,$$

where λ is a bifurcation parameter, and the continuation method allows the tracing of solution branches beyond turning points.

To validate the method, the authors applied it to classical nonlinear PDEs such as the viscous Burgers equation,

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0,$$

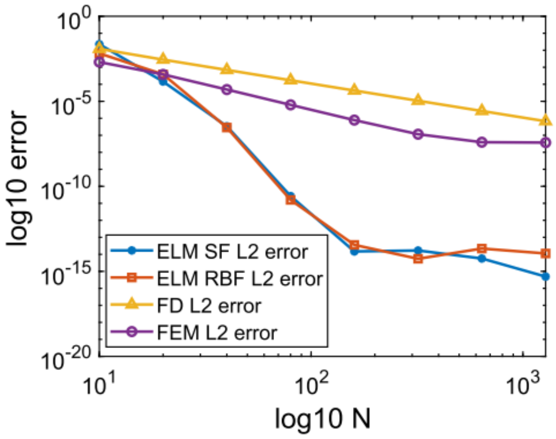

Figure 2 Numerical solution and accuracy of the FD, FEM and the proposed machine learning (ELM) schemes for the one-dimensional viscous Burgers problem with Dirichlet boundary conditions (Fabiani, Calabrò and Russo).

b. Physics-Informed Neural Networks (PINN)

Physics-Informed Neural Networks (PINNs) represent a modern approach for solving partial differential equations (PDEs) by embedding the governing physical laws directly into the loss function of a neural network. In this approach, a neural network is trained to approximate the solution function $u(x,t)$ across the spatial and temporal domain $\Omega \times [0,T]$. Unlike traditional numerical methods, PINNs do not require mesh generation or explicit discretization of the domain.

Consider a general nonlinear PDE of the form

$$N[u(x,t)] = f(x,t), \quad x \in \Omega, \quad t \in [0,T],$$

with boundary and initial conditions:

$$u(x,t) = g(x,t), \quad x \in \partial\Omega \; or \; t = 0.$$

In the PINN framework, the neural network $u_\theta(x,t)$ is parameterized by θ is trained such that the PDE residual

$$R(x,t;\theta) = N[u_\theta(x,t)] - f(x,t)$$

is minimized at a set of collocation points $\{x_i, t_i\}$ inside the domain. The boundary and initial conditions are enforced through additional terms in the loss function, leading to the total loss :

$$L(\theta) = \frac{1}{N_r}\sum_{i=1}^{N_r} |R(x_i,t_i;\theta)|^2 + \frac{1}{N_b}\sum_{j=1}^{N_b} \left|u_\theta(x_j,t_j) - g(x_j,t_j)\right|^2$$

where N_r and N_b are the numbers of residual and boundary/initial points, respectively (Raissi, Perdikaris and Karniadakis).

The procedure for applying PINNs to solve nonlinear partial differential equations can be summarized in the following steps:

1. **Problem Formulation:** Define the governing PDE along with the initial and boundary conditions.
2. **Neural Network Construction:** Design a neural network $u_\theta(x,t)$ with trainable parameters θ, where inputs are the spatial and temporal coordinates and the output is the approximate solution.
3. **Collocation Point Sampling:** Select collocation points inside the domain, on the boundary, and along the initial condition surface.

4. **Residual Definition:** Compute the PDE residual at collocation points:

5. **Loss Function Construction:** Define the total loss as the weighted sum of PDE residual loss, boundary condition loss, and initial condition loss:

$$L = L_{PDE} + L_{IC} + L_{BC}.$$

6. **Training:** Optimize the network parameters θ using gradient-based optimizers (e.g., Adam, L-BFGS) with automatic differentiation for derivatives.

7. **Validation:** Assess the accuracy by comparing against analytical solutions (if available) or high-fidelity numerical benchmarks, and check convergence through residual analysis.

c. Fourier Neural Operators

The problem with PINN algorithm lies in its dependance on the initial conditions, and boundary conditions, which makes it harder to predict large-scale simulations. This limitation inspired a new method for solving PDEs, called Neural Operators (NO).

At their core, Neural Operators transform an input function $a(x)$ to an output function $u(x)$. It is important to notice that, differently from classical Neural Networks, Neural operators work directly in the functional space, meaning their inputs and outputs are functions, not just vectors. This allows them to handle complex, continuous data more naturally.

Formally, a Neural Operator seeks to approximate an operator:

$$G: a(x) \mapsto u(x)$$

where $a(x)$ is an input function (such as forcing terms or coefficients) and $u(x)$ is the solution function of a PDE. The architecture is generally composed of three stages:

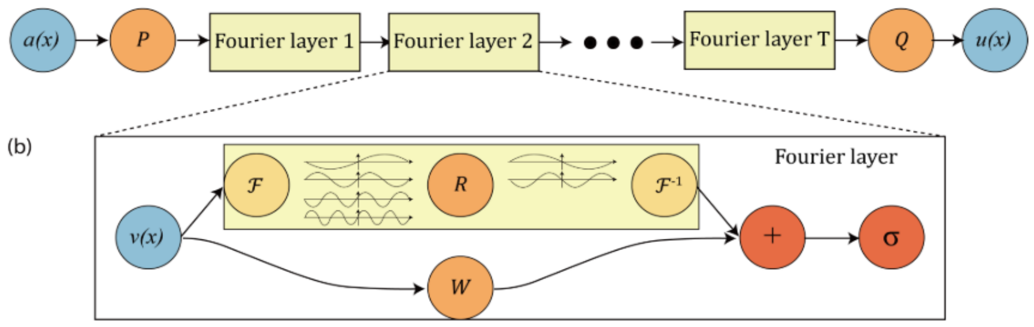

Figure 3 The full architecture of neural operator

1. **Encoder:** projects the input function into an embedding space suitable for processing.

2. **Processor:** applies multiple layers of kernel integral operators combined with nonlinear activations to capture local and general interactions of the data.

3. **Decoder:** maps the processed embedding back into the functional output space, which is considered our predicted output.

The **Fourier Neural Operator (FNO)** uses the Fourier transform to perform global convolutions efficiently, capturing long-range dependencies in PDE solutions (Li, Kovachki and Azizzadenesheli) . By working directly in functional spaces, Neural Operators achieve a balance of efficiency, accuracy, and generalization.

5. CONCLUSION

In this paper, we have discussed novel numerical methods for solving nonlinear differential equations. We focused on three main algorithms: Extreme Learning Machines, Physics-Informed Neural Networks, and Fourier Neural Operators. We found that these methods can solve a wide range of nonlinear differential equations, especially partial differential equations such as Burger's equation and Navier-Stokes. First, we laid the background, discussing the various methods of solving differential equations in general, including Newton-Raphson, Finite Differences, and other methods. Then, we mentioned the limitations of these methods brieflly, and found that most of the problems exhibits some shortcomings that can be mitigated. This research reviewed both classical and modern approaches for solving nonlinear differential equations. We began by recalling the basic ideas of stability, convergence, and stiffness, which shape how numerical methods perform in practice. Classical schemes such as Euler and Runge–Kutta were discussed for their simplicity, but also for their well-known difficulties in handling nonlinear or stiff problems. Extreme Learning Machines (ELMs) were shown to be fast and effective for bifurcation analysis, while Physics-Informed Neural Networks (PINNs) offered a flexible way to incorporate physical laws directly into the training process, and Fourier Neural Operators (FNOs) provided a scalable approach for learning solution operators across whole families of PDEs. Each method comes with its own strengths, making it suitable for different applications.

6. REFERENCES

[1] Atkinson, Kendall E. *An Introduction to Numerical Analysis*. 2nd. Wiley, 2008.
[2] Baty, Hubert. "A hands-on introduction to Physics-Informed Neural Networks for solving partial differential equations with benchmark tests taken from astrophysics and plasma physics." Tech. rep. 2024.
[3] Boyd, John P. *Chebyshev and Fourier Spectral Methods*. 2nd. Dover Publications, 2001.
[4] Butcher, John C. *Numerical Methods for Ordinary Differential Equations*. 3rd. Wiley, 2016.
—. *Numerical Methods for Ordinary Differential Equations*. John Wiley & Sons, 2016.
[5] Epstein, Irving R. and John A. Pojman. *An Introduction to Nonlinear Chemical Dynamics: Oscillations, Waves, Patterns, and Chaos*. New York: Oxford University Press, 1998.
[6] Fabiani, Gianluca, et al. "Numerical solution and bifurcation analysis of nonlinear partial differential equations with extreme learning machines." *Journal of Scientific Computing* 89.2 (2021).
[7] Hairer, Ernst and Gerhard Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer, 1996.
—. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. 2nd. Springer, 1996.
[8] Hairer, Ernst, Christian Lubich and Gerhard Wanner. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. 2nd. Vol. 31. Springer, 2006.
[9] Hairer, Ernst, Syvert P. Nørsett and Gerhard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 2nd. Springer, 1993.
—. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Springer, 1993.
[10] Hochbruck, Marlis and Alexander Ostermann. "Exponential integrators." *Acta Numerica* 19 (2010): 209–286.
[11] Hundsdorfer, Willem and Jan G. Verwer. *Numerical Solution of Time-Dependent Advection-Diffusion-Reaction Equations*. Vol. 33. Springer, 2003.
[12] LeVeque, Randall J. *Finite Difference Methods for Ordinary and Partial Differential Equations: Steady-State and Time-Dependent Problems*. SIAM, 2007.
[13] Li, Zongyi, et al. "Fourier Neural Operator for Parametric Partial Differential Equations." (2021). <http://arxiv.org/abs/2010.08895>.
[14] Lorenz, Edward N. "Deterministic Nonperiodic Flow." *Journal of the Atmospheric Sciences* 20 (1963): 130–141.
[15] Morton, K. W. and D. F. Mayers. *Numerical Solution of Partial Differential Equations*. Cambridge University Press, 2005.
[16] Murray, J. D. *Mathematical Biology I: An Introduction*. 3rd. Vol. 17. Springer, 2002.
[17] Nayfeh, Ali H. and Dean T. Mook. *Nonlinear Oscillations*. New York: Wiley, 1973.
[18] Quarteroni, Alfio, Riccardo Sacco and Fausto Saleri. *Numerical Mathematics*. 2nd. Springer, 2007.
[19] Raissi, M., P. Perdikaris and G. E. Karniadakis. "Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations." Tech. rep. 2018. <https://www.sciencedirect.com/science/article/pii/S0021999118307125>.

[20] Raissi, Maziar, Paris Perdikaris and George E. Karniadakis. “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.” *Journal of Computational Physics* 378 (2019): 686–707.
[21] Remani, Courtney. “Numerical Methods for Solving Systems of Nonlinear Equations.” Tech. rep. 2012.
[22] Söderlind, Gustaf. “Time-step selection algorithms: Adaptivity, control, and signal processing.” *Applied Numerical Mathematics* 56 (2006): 488–502.
[23] Strogatz, Steven H. *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry, and Engineering*. 2nd. CRC Press, 2018.
[24] Thompson, David B. and David B. Thompson. “Numerical Methods 101 - Convergence of Numerical Models.” Tech. rep. 1992
. <https://digitalcommons.unl.edu/usgsstaffpubhttps://digitalcommons.unl.edu/usgsstaffpub/115>.
[25] Trefethen, Lloyd N. and I. I. I. David Bau. *Numerical Linear Algebra*. SIAM, 1996.
[26] Trefethen, Lloyd N. *Spectral Methods in MATLAB*. SIAM, 2000.