JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

APPLICATION OF MACHINE LEARNING IN MATHEMATICAL PROBLEM SOLVING: EXPERIMENTAL RESULTS, ANALYSIS AND INTERPRETATION

Sumit Kumar Srivastava[1] Amit Kumar Pandey[2] Avneesh Kumar[3]
1) Research Scholar , Dr. K.N.Modi University ,Newai Rajasthan
2) Assitant Professor , Sharadchandra Arts Commerce & Science College Naigaon , Dist-Nanded
3) Assitant Professor, Dr. K.N.Modi University ,Newai Rajasthan

ABSTRACT

The growing integration of machine learning with mathematical computation has introduced new approaches for approximating and solving complex problems. This study investigates its applicability to nonlinear algebraic equations, ordinary differential equations, nonlinear function approximation, and mathematical optimization. Conventional procedures are compared with supervised and residual-based neural-network models in terms of accuracy, convergence, generalization, and computational relevance.

For nonlinear equations, Newton-Raphson iteration produced a highly accurate solution for an individual cubic, whereas a multilayer perceptron learned a reusable coefficient-to-root mapping for 2,000 depressed cubic equations. The model achieved a test MAE of 0.008862, RMSE of 0.025214, and R^2 of 0.999538. For the initial-value problem $y' = y$, $y(0) = 1$, a residual-based neural trial solution closely reproduced the exact solution $y = e^x$, with an RMSE of 7.19×10^{-9}. By comparison, Euler's method with $h = 0.1$ produced an absolute error of approximately 0.124539 at $x = 1$. A supervised neural network approximating $\sin(x)$ over $[0, 2\pi]$ achieved a test MAE of 0.002098, RMSE of 0.002529, and R^2 of 0.999988. An analytical optimization benchmark was additionally used to formulate a verifiable framework for surrogate-based optimization.

The results show that machine-learning models can provide accurate and reusable approximations when mathematical tasks involve repeated evaluations, nonlinear mappings, or governing differential constraints. Their reliability, however, depends on the data, architecture, optimization procedure, problem domain, and validation criteria. Conventional methods remain preferable for many simple and well-structured problems because of their precision, transparency, and theoretical guarantees. Machine learning is therefore most appropriately viewed as a complementary computational methodology rather than a universal replacement for established analytical and numerical techniques.

Keywords: Machine Learning; Mathematical Problem Solving; Artificial Neural Networks; Nonlinear Algebraic Equations; Ordinary Differential Equations; Function Approximation; Numerical Analysis; Surrogate Optimization.

Introduction

This paper integrates the mathematical formulation, computational implementation, experimental results, and interpretation of the selected problems considered in the present study. Instead of separating the application framework from the numerical results, each experiment is presented as a complete sequence: mathematical problem, conventional solution, machine-learning formulation, experimental procedure, observed result, error analysis, and interpretation. This organization provides a direct connection between the mathematical theory and the computational evidence. Rigorous error analysis of physics-informed models has increasingly focused on separating approximation, generalization, and optimization contributions rather than treating training loss as the total numerical error [8–11].

The central purpose of the paper is to investigate how machine-learning techniques can be used in solving or approximating selected mathematical problems and to compare their behaviour with established analytical or numerical procedures. The investigation does not begin with the assumption that a learning model must outperform a traditional method. The comparison is based on mathematical accuracy, convergence, generalization, computational requirements, and the nature of the problem being solved. Theoretical studies of physics-informed learning caution that convergence of the training objective and convergence of the computed mathematical solution are related but distinct questions [8–11].

Four representative areas are considered: nonlinear algebraic equations, ordinary differential equations, mathematical function approximation, and optimization. These areas were selected because they demonstrate different relationships between mathematics and machine learning. In some cases, machine learning is used to approximate a mapping generated by a conventional solver. In another case, the governing differential equation is incorporated directly into the learning objective. The optimization example illustrates the concept of a learned surrogate for an objective function. Software frameworks such as DeepXDE illustrate how residual-based learning, automatic differentiation, adaptive sampling, and forward/inverse differential-equation problems can be implemented within a unified workflow [7,17].

Overall Experimental and Mathematical Framework

Let x denote an input variable and y the corresponding exact or reference output. A general mathematical relationship can be represented as

$$y = F(x) \quad \ldots\ldots(1.1)$$

For N observations, the available information consists of pairs (x_i, y_i), i = 1, 2, …, N. A learning model with trainable parameters θ produces

$$\hat{y}_i = M(x_i;\, \theta) \quad \ldots\ldots(1.2)$$

The prediction error is defined by

$$e_i = y_i - \hat{y}_i \quad \ldots\ldots(1.3)$$

The training process seeks parameter values for which an appropriate loss function is minimized. For regression, a common choice is

$$L(\theta) = (1/N)\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad \text{.......} (1.4)$$

The exact form of the learning problem differs among the experiments. For root prediction, the inputs are equation coefficients and the output is a real root. For the differential equation, the loss is constructed from the residual of the governing equation. For function approximation, sampled values of a known nonlinear function are used. In optimization, a model may approximate an objective function that would otherwise require repeated evaluation. [3,4,12,19,23–27]

Performance Evaluation Criteria

A mathematical learning model should not be judged by a single statistic. The present study therefore uses complementary error measures. The absolute error for an observation is

$$AE_i = |y_i - \hat{y}_i| \quad \text{.......} (1.5)$$

The Mean Absolute Error is

$$MAE = (1/N)\sum_{i=1}^{N}|y_i - \hat{y}_i| \quad \text{.......} (1.6)$$

The Mean Squared Error and Root Mean Squared Error are

$$MSE = (1/N)\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \quad \text{.......} (1.7)$$

$$RMSE = \sqrt{[(1/N)\sum_{i=1}^{N}(y_i - \hat{y}_i)^2]} \quad \text{.......} (1.8)$$

For regression experiments, the coefficient of determination is

$$R^2 = 1 - \left[\sum_{i=1}^{N}(y_i - \hat{y}_i)^2\right] / \left[\sum_{i=1}^{N}(y_i - \bar{y})^2\right] \quad \text{.......} (1.9)$$

where $\bar{y}$ is the arithmetic mean of the reference outputs. Maximum absolute error is also considered because a small average error can conceal an isolated prediction with a much larger deviation. [18,19]

$$E_{max} = max\,|y_i - \hat{y}_i| \quad \text{.......} (1.10)$$

These measures are interpreted in the context of the mathematical problem rather than mechanically. For example, an error that is negligible for one application may be unacceptable for another application requiring high numerical precision.

Experiment I: Nonlinear Algebraic Equation

Mathematical Problem

The first benchmark problem is the nonlinear algebraic equation

$$x^3 - x - 2 = 0 \quad \text{.......} (1.11)$$

Let $f(x) = x^3 - x - 2$ ....(1.12)

At x = 1, f(1) = −2, while at x = 2, f(2) = 4. Since the polynomial is continuous and changes sign between these points, at least one real root exists in the interval (1, 2).

Conventional Solution by Newton–Raphson Method

The Newton–Raphson method generates successive approximations according to

$$x_{n+1} = x_n V - f(x_n)/f'(x_n) \quad \text{.....(1.13)}$$

For the selected polynomial,

$$f'(x) = 3x^2 - 1 \quad \text{.....(1.14)}$$

and hence

$$x_{n+1} = x_n - (x_n{}^3 - x_n - 2)/(3x_n{}^2 - 1) \quad \text{.....(1.15)}$$

Using the initial approximation $x_0 = 1.5$ gives the following sequence.

Iteration n	Approximation x_n	Successive change
0	1.5000000000	—
1	1.5217391304	0.0217391304
2	1.5213798060	0.0003593244
3	1.5213797068	0.0000000992

The iteration stabilizes rapidly and gives

$$x \approx 1.5213797068 \quad \text{..... (1.16)}$$

This calculation demonstrates the strength of a conventional numerical method for a single well-behaved nonlinear equation. It also provides a reliable reference against which a learned approximation can be evaluated. [1,20,21]

Why a Parameterized Learning Problem is Required

Training a neural network using only one equation and one corresponding root would not constitute a meaningful machine-learning experiment. A learning model requires a collection of related input-output examples. The problem is therefore generalized to the depressed cubic family

$$x^3 + ax + b = 0 \quad \text{.....(1.17)}$$

For each coefficient pair (a, b), the target is a selected real root r. The learning task become

$$(a, b) \rightarrow r \quad \text{.....(1.18)}$$

and the predicted root is represented by

$$\hat{r} = M(a, b; \theta) \quad \text{.....(1.19)}$$

Root Selection and Dataset Construction

A depressed cubic can have either one real root or three real roots. Its discriminant is

$$\Delta = -4a^3 - 27b^2 \quad \text{.....(1.20)}$$

For the present single-output regression experiment, only coefficient pairs satisfying $\Delta < 0$ were retained. This condition ensures one real root and avoids ambiguity in the target mapping. Two thousand admissible coefficient pairs were generated from the coefficient interval $[-3, 3]$. Numerical polynomial roots were used to construct the reference targets. [1,20,21,30,31]

A reproducible random permutation with seed 42 divided the 2,000 equations into 1,600 training cases and 400 independent test cases. The test equations were not used for model fitting.

Neural-Network Model and Training

The coefficient pair (a, b) formed the two-dimensional input. The output was the corresponding real root. A multilayer perceptron with two hidden layers containing 60 hyperbolic-tangent units each was fitted using the L-BFGS optimization procedure. The regression loss was based on squared root-prediction error. [12–16,22,27–29]

$$L_{\mathrm{r}}(\theta) = (1/N_{\mathrm{tr}})\, \Sigma\, [r_{\mathrm{i}} - M(a_{\mathrm{i}}, b_{\mathrm{i}};\, \theta)]^2 \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1.21)$$

Experimental Results

Performance measure	Observed value
Total equations	2,000
Training equations	1,600
Test equations	400
MAE	0.008862
RMSE	0.025214
Maximum absolute error	0.253089
R^2	0.999538
Optimization iterations	337

The model achieved a small average error across the independent test equations. The R^2 value indicates that the model reproduced most of the variation in the selected roots. However, the maximum absolute error was considerably larger than the MAE. This observation is important because mathematical reliability should not be inferred from an average statistic alone.

The root-prediction experiment therefore demonstrates both the potential and the limitation of a learned mapping. It can produce rapid approximations for previously unseen coefficient pairs within the sampled domain, but individual predictions still require validation when high precision is essential. Recent reviews also identify scalability, loss balancing, sampling, reproducibility, and fair benchmarking against classical solvers as continuing challenges [9–11].

Newton–Raphson and Machine-Learning Interpretation

Newton–Raphson and the neural model solve different computational tasks in this experiment. Newton–Raphson directly solves an individual equation. The neural model incurs an initial training cost but subsequently predicts roots across a family of equations. Consequently, it would be misleading to state simply that one method is better than the other.

For an isolated equation such as $x^3 - x - 2 = 0$, Newton–Raphson is inexpensive, transparent, and highly accurate. A learned model becomes more relevant when many related equations must be evaluated repeatedly and a small approximation error is acceptable. The experiment therefore supports a problem-dependent interpretation of computational efficiency.

Additional Nonlinear-Equation Benchmark Examples

Three further equations are included within the same nonlinear-equation experiment to broaden the benchmark set. Each is evaluated with Newton–Raphson iteration and residual verification. These verified roots can subsequently serve as reference targets for a learning-based root predictor; no additional neural accuracy is claimed without an actual trained output.

Example A: $x^3 - 2x - 5 = 0$

Iteration	x_n	$f(x_n)$
0	2.0000000000	1.000e+00
1	2.1000000000	6.100e-02
2	2.0945681211	1.857e-04
3	2.0945514817	1.740e-09
4	2.0945514815	8.882e-16
5	2.0945514815	8.882e-16

Example B: $x^2 - 3 = 0$

Iteration	x_n	$f(x_n)$
0	2.0000000000	1.000e+00
1	1.7500000000	6.250e-02
2	1.7321428571	3.189e-04
3	1.7320508100	8.473e-09
4	1.7320508076	4.441e-16

Example C: $\cos(x) - x = 0$

Iteration	x_n	$f(x_n)$
0	0.7000000000	6.484e-02
1	0.7394364978	5.881e-04
2	0.7390851605	4.561e-08
3	0.7390851332	2.220e-16
4	0.7390851332	0.000e+00

The three added problems cover a cubic equation, a quadratic irrational root, and a transcendental fixed-point equation. For learning-based comparison, both distance from the verified root and substitution residual should be reported.

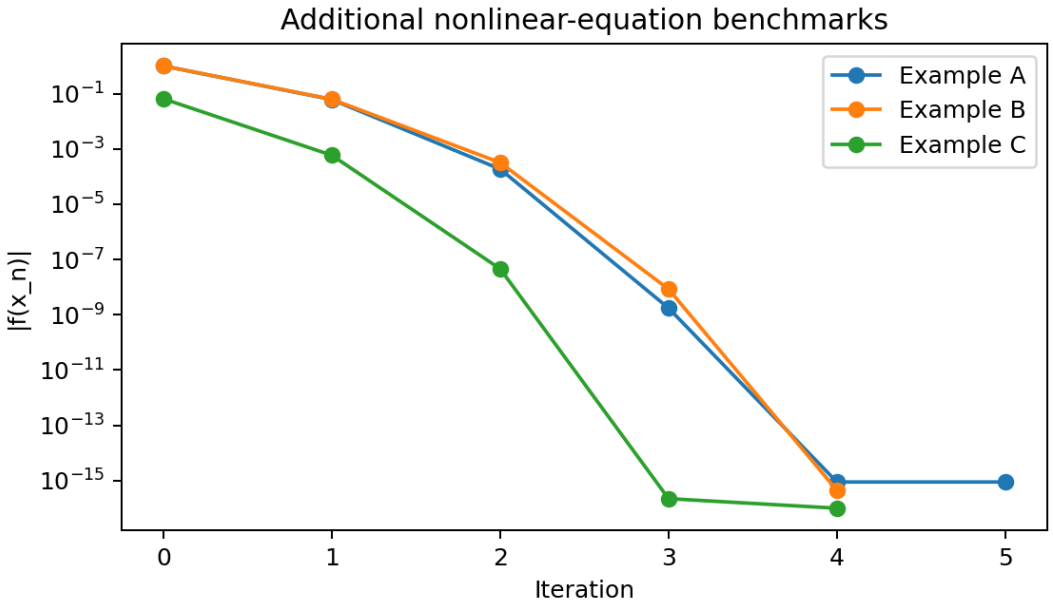

Figure 1: Residual convergence for three additional nonlinear-equation examples.

Experiment II: Ordinary Differential Equation

Mathematical Formulation and Exact Solution

The second experiment considers the initial-value problem

$$dy/dx = y, \quad y(0) = 1 \quad \text{.....(1.22)}$$

Separating the variables gives

$$dy/y = dx \quad \text{..... (1.23)}$$

Integration yields

$$ln|y| = x + C \quad \text{.....(1.24)}$$

and therefore

$$y = Ce^{x} \quad \text{..... (1.25)}$$

Applying the initial condition gives C = 1, so the exact solution is

$$y(x) = e^{x} \quad \text{.....(1.26)}$$

The availability of a closed-form solution allows every numerical or neural approximation to be checked directly against exact values.

Euler Method as a Numerical Benchmark

For $dy/dx = f(x,y)$, Euler's method is

$$y_{n+1} = y_n + h\, f(x_n, y_n) \quad \text{..... (1.27)}$$

Since $f(x,y) = y$,

$$y_{n+1} = (1 + h)y_n \quad \text{.....(1.28)}$$

With h = 0.1, this becomes $y_{n+1} = 1.1y_n$. Starting from $y_0 = 1$ gives the following comparison.

x	Exact e^x	Euler approximation	Absolute error
0.0	1.000000	1.000000	0.000000
0.1	1.105171	1.100000	0.005171
0.2	1.221403	1.210000	0.011403
0.3	1.349859	1.331000	0.018859
0.4	1.491825	1.464100	0.027725
0.5	1.648721	1.610510	0.038211
0.6	1.822119	1.771561	0.050558
0.7	2.013753	1.948717	0.065036
0.8	2.225541	2.143589	0.081952
0.9	2.459603	2.357948	0.101655
1.0	2.718282	2.593742	0.124539

At x = 1, the Euler approximation is approximately 2.593742, whereas the exact value is approximately 2.718282. The absolute error is about 0.124540 and the relative error is approximately 4.58%. The error increases through the interval because local discretization errors accumulate from one Euler step to the next.

This result must be interpreted with the numerical configuration in mind. Euler's method is first order and h = 0.1 is a relatively coarse step. A smaller step or a higher-order method would reduce the error. The purpose of this benchmark is therefore to provide a transparent baseline rather than to represent the best available conventional ODE solver.

Neural Trial Solution

For the learning-based formulation, let N(x; θ) denote the scalar output of a neural network. The trial solution is defined as

$$y_t(x;\theta) = 1 + xN(x;\theta) \quad \text{.....(1.29)}$$

This construction automatically satisfies the initial condition because $y_t(0; \theta) = 1$ for every possible parameter vector θ. Differentiating by the product rule gives

$$dy_t/dx = N(x;\theta) + x\,\partial N(x;\theta)/\partial x \quad \text{.....(1.30)}$$

The differential equation may be written as dy/dx − y = 0. The neural residual is therefore

$$R(x;\theta) = dy_t/dx - y_t(x;\theta) \quad \text{.....(1.31)}$$

Substituting the trial solution gives

$$R(x;\theta) = N(x;\theta) + x\,\partial N(x;\theta)/\partial x - 1 - xN(x;\theta) \quad \text{.....(1.32)}$$

For N collocation points, the residual-based objective is

$$L^{\mathrm{ODE}}(\theta) = (1/N)\sum_{i=1}^{N_C}[R(x_i;\theta)]^2 \quad \text{.....(1.33)}$$

This formulation differs from ordinary supervised regression. Exact e^x values are not required as training labels. Instead, the governing differential equation itself supplies the condition that the learned function is required to satisfy. The construction of trial solutions that satisfy initial/boundary conditions by design follows the neural differential-equation approach of Lagaris, Likas and Fotiadis . [2]

Neural Experimental Configuration

The computational experiment used 101 equally spaced collocation points over the interval [0, 1]. A single hidden layer with ten tanh units was used in the trial network. Parameter initialization used random seed 42. Nonlinear least-squares optimization was allowed up to 20,000 function evaluations. The optimization reached this evaluation limit, and this fact is retained explicitly in the reporting rather than treating the run as proof of optimizer convergence. Residual-based neural formulations of differential equations are closely related to the broader physics-informed learning framework . Modern scientific-machine-learning frameworks further emphasize explicit enforcement of governing equations, initial/boundary conditions, and residual-based verification [2,3,7,11].

Neural ODE Results

x	Exact solution	Neural approximation	Absolute error
0.0	1.000000000	1.000000000	0.00×10^{0}
0.1	1.105170918	1.105170921	3.13×10^{-9}
0.2	1.221402758	1.221402770	1.23×10^{-8}
0.3	1.349858808	1.349858820	1.19×10^{-8}
0.4	1.491824698	1.491824704	6.38×10^{-9}
0.5	1.648721271	1.648721267	3.41×10^{-9}
0.6	1.822118800	1.822118797	3.40×10^{-9}
0.7	2.013752707	2.013752703	4.45×10^{-9}
0.8	2.225540928	2.225540926	1.63×10^{-9}
0.9	2.459603111	2.459603115	4.18×10^{-9}
1.0	2.718281828	2.718281836	7.05×10^{-9}

Error measure	Observed value
MAE	5.50×10^{-9}
RMSE	7.19×10^{-9}
Maximum absolute error	1.24×10^{-8}
Mean squared residual at 101 collocation points	1.61×10^{-14}
Function-evaluation limit	20,000; limit reached

The neural approximation follows the analytical solution extremely closely on this elementary benchmark. The small residual indicates that the fitted trial function nearly satisfies the governing differential equation at the selected collocation points. Nevertheless, the optimizer reaching its evaluation limit is an important methodological detail and prevents an unsupported claim that the optimization itself fully converged.

The result should be interpreted as a controlled proof of concept. The equation $y' = y$ is smooth, one-dimensional, non-stiff, and has a simple exact solution. Comparable accuracy cannot be assumed for nonlinear systems, stiff equations, long time intervals, or high-dimensional partial differential equations. Recent

numerical-analysis literature also distinguishes approximation, generalization, and training errors in physics-informed learning, emphasizing that optimization error can be a major practical bottleneck [8–10].

Exact, Euler and Neural Comparison

Method	Approximation at x = 1	Absolute error at x = 1
Exact analytical solution	2.718281828	0
Euler method, h = 0.1	2.593742460	1.245394×10^{-1}
Residual-based neural model	2.718281836	7.05×10^{-9}

Under the stated settings, the residual-based neural result is much closer to the analytical solution than the coarse Euler result. This numerical difference must not be generalized into a claim that neural networks are universally superior to conventional ODE methods. A higher-order Runge–Kutta solver, adaptive step-size method, or smaller Euler step could change the conventional comparison substantially.

The important research observation is instead that mathematical constraints can be incorporated directly into a neural objective. In this experiment, the initial condition is enforced by the trial solution and the differential equation is enforced through the residual loss. Thus, mathematical structure and machine learning operate together.

Additional ODE Benchmark Examples

Three further initial-value problems are included within the ODE experiment. Exact analytical solutions are available for all three, while Euler's method with h = 0.1 provides a reproducible conventional baseline. The same test-grid, residual, initial-condition, and pointwise-error checks used in the main experiment can therefore be applied to any trained neural solution. [1,20,21]

Example A: $y' = -2y$, $y(0)=1$

x	Exact	Euler	Absolute error
0.0	1.00000000	1.00000000	0.00000000
0.2	0.67032005	0.64000000	0.03032005
0.4	0.44932896	0.40960000	0.03972896
0.6	0.30119421	0.26214400	0.03905021
0.8	0.20189652	0.16777216	0.03412436
1.0	0.13533528	0.10737418	0.02796110

Example B: $y' = y$, $y(0)=1$

x	Exact	Euler	Absolute error
0.0	1.00000000	1.00000000	0.00000000
0.2	1.22140276	1.21000000	0.01140276
0.4	1.49182470	1.46410000	0.02772470
0.6	1.82211880	1.77156100	0.05055780
0.8	2.22554093	2.14358881	0.08195212
1.0	2.71828183	2.59374246	0.12453937

Example C: $y' = x + y$, $y(0)=1$

x	Exact	Euler	Absolute error
0.0	1.00000000	1.00000000	0.00000000
0.2	1.24280552	1.22000000	0.02280552
0.4	1.58364940	1.52820000	0.05544940
0.6	2.04423760	1.94312200	0.10111560
0.8	2.65108186	2.48717762	0.16390424
1.0	3.43656366	3.18748492	0.24907874

The added ODEs represent decay, growth, and a non-homogeneous equation. This widens the numerical behavior represented in the paper without changing the original experimental conclusions.

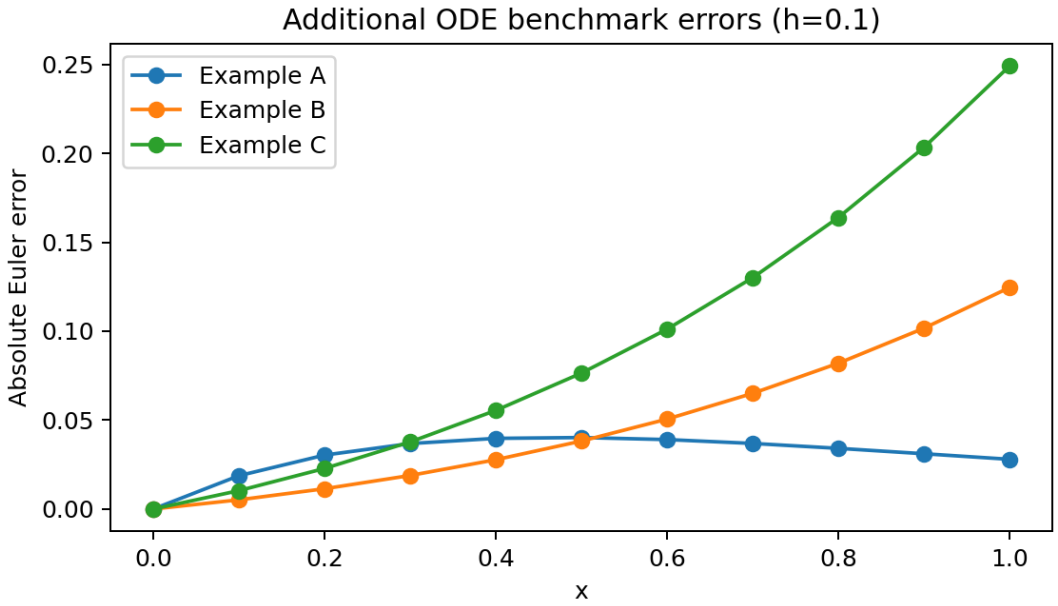

Figure 2 : Euler absolute-error profiles for three additional ODE examples.

Experiment III: Mathematical Function Approximation

Problem Definition

The third experiment investigates supervised approximation of the nonlinear function

$$y = \sin x, \quad 0 \le x \le 2\pi \qquad \text{.....(1.34)}$$

Unlike the ODE experiment, this is a conventional supervised-learning problem. Exact pairs (x_i, sin x_i) are available and can be divided into training and independent test subsets.

Dataset and Model Configuration

A total of 401 equally spaced observations were generated over [0, 2π]. A reproducible random permutation with seed 42 assigned 321 observations to training and 80 observations to testing. The test observations were not used during model fitting.

The neural model consisted of two hidden layers containing 40 tanh units each. L-BFGS optimization was used with regularization parameter 10^{-6} and a maximum of 5,000 iterations. The model learned the mapping x → sin x through squared prediction error. [4–6,13,22]

Test Results

Performance measure	Observed value
Total observations	401
Training observations	321
Test observations	80
MAE	0.002098
RMSE	0.002529
Maximum absolute error	0.005206
R^2	0.999988

The low test errors indicate that the network approximated the smooth sine relationship accurately at observations not used for training. The R^2 value is close to unity, but the interpretation is strengthened by reporting the error magnitudes themselves rather than relying on R^2 alone.

The purpose of this experiment is not to suggest that a neural network should replace direct evaluation of the sine function. The exact function is already known and can be evaluated efficiently. Instead, the experiment provides a controlled demonstration of nonlinear function learning and generalization within a known domain.

Selected-Point Interpretation

Several mathematically significant points provide an intuitive check of the learned approximation. At $x = 0$, $\pi/2$, π, $3\pi/2$, and 2π, the exact function values are 0, 1, 0, −1, and 0 respectively. The learned model remains close to these values, with deviations consistent with the small test-set error measures.

Because the model is trained from finite samples, its output is an approximation rather than a symbolic identity. In particular, accurate interpolation within $[0, 2\pi]$ does not imply that the same network will preserve periodic behaviour indefinitely outside the training interval.

Training, Testing and Generalization

A central requirement in machine-learning-based mathematics is separation between model fitting and final evaluation. If the same observations are used for both purposes, a small error may reflect memorization rather than generalization. The independent test subset used here provides evidence that the learned mapping extends to unseen points within the sampled interval. Standard machine-learning methodology similarly distinguishes training, validation and test performance to assess generalization [12,18,19].

The distinction between interpolation and extrapolation is also important. Test points inside the training domain assess interpolation. Predictions outside $[0, 2\pi]$ would constitute extrapolation and should be treated as a separate experiment. The present findings are therefore restricted to the investigated domain. [12,19]

Additional Function-Approximation Benchmark Examples

Three further exact target functions are included within the function-approximation experiment: x^2, e^x, and $\cos(x)$ on the common interval $[-1,1]$. Their exact values provide unambiguous reference data for MAE, RMSE, maximum error, and R^2 after a model is trained. No unrecorded neural predictions have been inserted.

x	x^2	e^x	cos(x)
-1.0	1.000000	0.367879	0.540302
-0.5	0.250000	0.606531	0.877583
0.0	0.000000	1.000000	1.000000
0.5	0.250000	1.648721	0.877583
1.0	1.000000	2.718282	0.540302

The polynomial, exponential, and trigonometric targets provide different curvature and shape characteristics. Training and test points should remain separated so that interpolation and generalization can be interpreted independently.

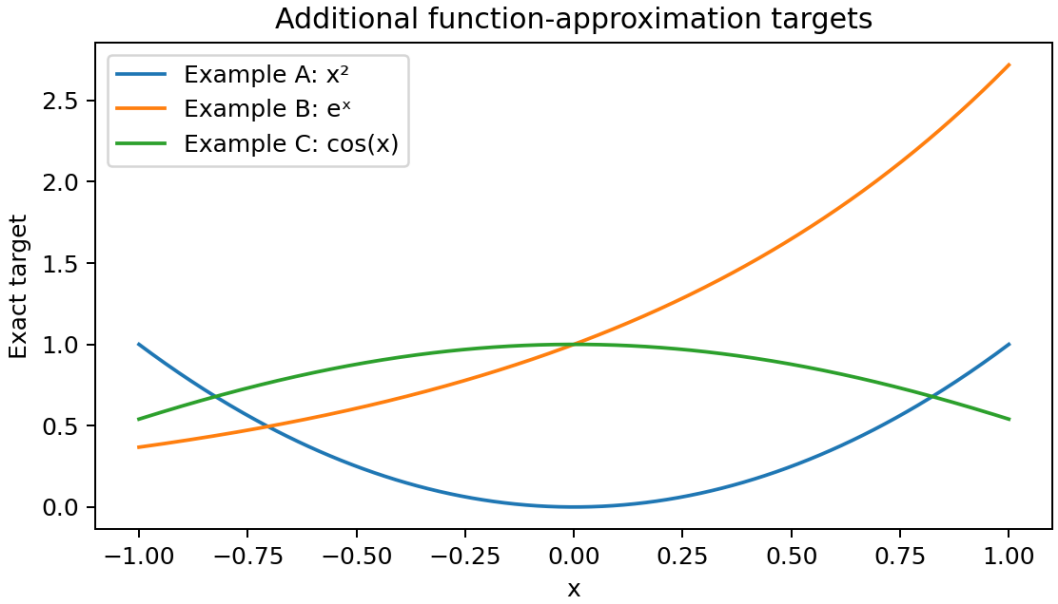

Figure 3 : Exact curves for three additional function-approximation examples.

Experiment IV: Mathematical Optimization

Analytical Benchmark

The optimization benchmark is

$$f(x) = (x - 3)^2 + 2 \quad \text{.....(1.35)}$$

Differentiation gives

$$f'(x) = 2(x - 3) \quad \text{..... (1.36)}$$

The stationary condition f′(x) = 0 yields

$$x^* = 3 \quad \text{..... (1.37)}$$

The second derivative is

$$f''(x) = 2 > 0 \quad \text{..... .....(1.38)}$$

Therefore x* = 3 is the unique global minimizer and

$$f(x^*) = 2 \quad \text{.....(1.39)}$$

Surrogate-Learning Formulation

For a computationally expensive objective function, a learning model can be used to construct a surrogate approximation

$$f(x) = M(x;\theta) \quad \text{………………………………………} (1.40)$$

The surrogate optimizer is

$$\hat{x}^* = \arg\min \hat{f}(x) \quad \text{……………………………………...} (1.41)$$

The location error and objective-value error can then be measured by

$$E_x = |x^* - \hat{x}^*| \quad \text{…………………………………………}(1.42)$$

$$E_f = |f(x^*) - f(\hat{x}^*)| \quad \text{……………………………………..}(1.43)$$

For the simple quadratic benchmark, direct analytical optimization is clearly preferable and no machine-learning model is needed in practice. The example is included to establish the mathematical structure required for surrogate optimization. A numerical surrogate result is intentionally not invented where a separately documented experiment has not been performed.

Additional Optimization Benchmark Examples

Three further one-dimensional convex objectives are included within the optimization experiment. Their analytical minima make them suitable for checking a trained surrogate or learning-based optimizer against a transparent mathematical reference. [23–27]

Example	Objective	Analytical minimizer	Minimum value
A	$f(x)=(x+2)^2+1$	$x^*=-2$	1
B	$f(x)=x^2+4x+7$	$x^*=-2$	3
C	$f(x)=(x-1)^2+2$	$x^*=1$	2

For a future trained surrogate, evaluation should include distance from the analytical minimizer, objective-value gap, feasibility where relevant, and computational cost. No surrogate optimum is reported here without an actual trained model output.

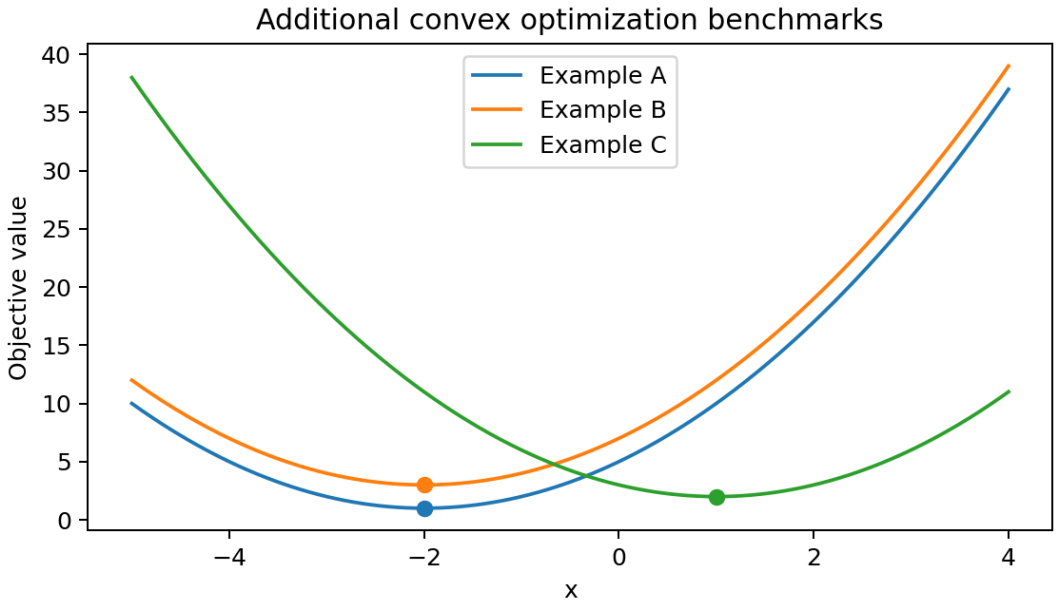

Figure 4 : Three additional convex optimization benchmarks and analytical minima.

Neural-Network Architecture and Mathematical Representation

A feedforward neural network transforms its input through successive affine and nonlinear operations. For hidden layer l, the transformation can be represented by Feedforward neural networks and their approximation properties are well established in the neural-network literature [4–6,13–15,28].

$$h^{(l)} = \varphi(W^{(l)}h^{(l-1)} + b^{(l)}) \quad \text{.....(1.44)}$$

where $W^{(l)}$ is the weight matrix, $b^{(l)}$ is the bias vector, and φ is the activation function. For scalar regression, the output layer can be written as [4–6,13,17]

$$\hat{y} = W^{(L)}h^{(L-1)} + b^{(L)} \quad \text{.....(1.45)}$$

The use of the tanh activation in the reported experiments is suitable for representing smooth nonlinear mappings. The architecture, however, is not mathematically unique. Different widths, depths, activation functions, and optimizers may produce different accuracy and convergence behaviour. For this reason, the architecture is treated as part of the experimental configuration and not as a universal optimal design. Smooth activation functions are particularly relevant when derivatives of the learned mapping are required .

Convergence Analysis

Convergence has different meanings for conventional numerical algorithms and machine-learning optimization. For Newton–Raphson iteration, convergence can be monitored through the successive difference

$$C_n = |x_{n+1} - x_n| \quad \text{.....(1.46)}$$

and a practical stopping condition is

$$|x_{n+1} - x_n| < \varepsilon \quad \text{.....(1.47)}$$

The root experiment shows a rapid decrease in successive changes from approximately 2.17×10^{-2} to 3.59×10^{-4} and then to approximately 9.92×10^{-8}.

For neural training, convergence is associated with reduction of the optimization objective. A stable low training loss is useful but is not equivalent to mathematical accuracy on unseen data. The supervised experiments must therefore be judged on independent test error. In the ODE experiment, the residual is additionally evaluated because the objective itself represents violation of the governing differential equation.

The ODE optimizer reached the stated function-evaluation limit. This distinction between a highly accurate achieved approximation and formal optimizer termination is important. Reporting it preserves the difference between numerical evidence and a stronger convergence claim.

Computational Efficiency Analysis

A conventional numerical method and a trained learning model incur computational cost in different ways. For K conventional solutions, total cost may be represented as Numerical algorithms and their error/convergence behaviour should be interpreted with respect to the method and computational configuration used [1,20,21,27].

$$T_{\mathrm{num}} = \Sigma_{\mathrm{k=1}}{}^{\mathrm{K}} T_{\mathrm{num,\,k}} \quad \text{……………………………… (1.48)}$$

For machine learning, total cost should include data generation, training, and subsequent predictions:

$$T^{\mathrm{ML}} = T_{data} + T_{train} + K\,T_{pred} \quad \text{…………………………(1.49)}$$

This expression explains why inference time alone is not an adequate measure of efficiency. A neural model may predict rapidly after training, but the initial generation of reference solutions and model optimization can be substantial. [22–27]

For a small number of simple problems, a conventional solver is often more economical. A learned surrogate becomes potentially attractive when K is large, the underlying solver is expensive, and the prediction error remains within the required tolerance. Thus, computational advantage depends on the number of repeated evaluations as well as on accuracy.

Overall Comparative Analysis

Problem	**Conventional / exact approach**	**Learning-based approach**	**Main observation**
Nonlinear equation	Newton–Raphson gives $x \approx 1.5213797068$	Parameterized cubic model: RMSE 0.025214	Conventional method is excellent for one equation; ML is reusable across a family.
Ordinary differential equation	Exact e^x; Euler h=0.1 has visible error	Residual neural approximation: RMSE $\approx 7.19 \times 10^{-9}$	Mathematical constraints can be embedded in learning.
Function approximation	Exact sin x is directly available	Supervised MLP: RMSE 0.002529; R^2 0.999988	Strong interpolation on unseen points in the sampled domain.

Optimization	Exact minimum x*=3, f=2	Surrogate framework formulated	ML is useful mainly when the original objective is expensive.

The comparison shows that the meaning of machine-learning usefulness changes across problem types. In the cubic experiment, the learned model approximates the output of a numerical solver across a parameter space. In the ODE experiment, the mathematical equation itself forms part of the learning objective. In the sine experiment, the network learns a known nonlinear mapping from labelled samples. In optimization, a surrogate would replace repeated expensive objective evaluations rather than replace the mathematical concept of minimization.

Detailed Discussion of Experimental Results

The nonlinear equation experiment confirms that established numerical algorithms remain highly effective for elementary problems. Newton–Raphson converged to high precision in only a few iterations. This result prevents an exaggerated interpretation of machine learning: there is little practical justification for training a network merely to solve one inexpensive cubic equation.

The parameterized root experiment provides a more meaningful machine-learning use case. The model learns a reusable mapping from coefficients to roots. Its low average error and high R^2 indicate strong predictive performance over the selected test distribution, while the larger maximum error shows that individual cases can still require verification. This balance between average performance and worst-case reliability is particularly important in mathematical computation.

The ODE experiment demonstrates a different form of integration between mathematics and machine learning. The initial condition is built into the trial function and the governing equation is represented by the residual. The network is therefore not merely fitting a table of exact solutions. The mathematical structure actively determines the objective being optimized.

The sine-function experiment provides a clean supervised-learning benchmark. Because the exact function is known, prediction errors can be measured without uncertainty. The experiment shows strong interpolation, but it also highlights a conceptual limitation: a learned approximation does not acquire the symbolic identity or global periodic guarantee of the exact sine function.

The optimization benchmark reinforces the same general principle. When an exact analytical solution is trivial, machine learning adds unnecessary complexity. Surrogate learning becomes scientifically interesting when evaluating the true objective is expensive, repeated many times, or embedded in a larger computational process.

Major Findings

- Conventional mathematical methods remain highly competitive for simple, well-defined problems with inexpensive analytical or numerical solutions.
- Newton–Raphson provided rapid and precise convergence for the selected nonlinear equation.
- A machine-learning model can learn a reusable coefficient-to-root mapping across a parameterized family of cubic equations, although individual prediction errors must still be monitored.

- Euler's method with h = 0.1 demonstrated the accumulation of discretization error and provided a transparent numerical baseline for the ODE experiment.
- A residual-based neural formulation can incorporate an initial condition and a differential equation directly into the learning framework.
- The residual neural model reproduced the elementary ODE benchmark with very small reported error, but the optimizer reached its function-evaluation limit and the result should not be generalized to difficult differential equations.
- The supervised sine experiment demonstrated accurate interpolation on unseen observations within the sampled interval.
- High R^2 alone is insufficient for mathematical validation; MAE, RMSE, maximum error, residual behaviour, and problem-specific tolerance should also be considered.
- Machine-learning inference speed should not be compared with a numerical solver without accounting for data-generation and training costs.
- The most defensible role of machine learning in this study is complementary: it can approximate reusable mappings or embed mathematical constraints when this provides a computational advantage.

Relationship of Findings to the Research Objectives

The experimental results directly address the objective of investigating the applicability of machine learning to mathematical problem solving. The experiments show that learning models can approximate nonlinear functions, parameterized equation solutions, and mathematically constrained differential-equation solutions under controlled conditions.

The results also address the objective of comparing learning-based approaches with conventional mathematical methods. The comparison reveals that superiority is not universal. Conventional methods possess advantages in transparency, theoretical structure, and direct precision for many elementary problems. Learning methods offer a different advantage when a trained approximation can be reused over many related inputs.

The investigation therefore refines the research question from a simple competition between traditional mathematics and machine learning to a more meaningful question: under what mathematical and computational conditions does a learned approximation provide sufficient accuracy and useful computational efficiency?

Practical Significance of Machine Learning in Mathematical Problems

The practical importance of machine learning is strongest when the mathematical task involves repeated evaluation, high-dimensional parameter spaces, computationally expensive simulation, or the need for rapid approximate predictions after an initial training stage. In such situations, a learned model may act as a surrogate for a slower numerical process. [23–27]

Machine learning can also be valuable when mathematical structure is incorporated into the model itself. The ODE experiment illustrates this possibility through a trial solution and residual loss. Such formulations preserve a direct connection between the governing mathematics and the learning procedure. This principle is central to neural differential-equation and physics-informed neural-network approaches . [2,3,7,11]

At the same time, the experiments demonstrate that exact formulas and reliable numerical algorithms should not be discarded merely because a machine-learning alternative exists. The choice of method should depend on accuracy requirements, computational cost, interpretability, repeatability, and the mathematical characteristics of the problem.

Reliability, Reproducibility and Experimental Discipline

Reproducibility is essential when machine-learning results are presented as part of a mathematical thesis. The reported experiments therefore specify dataset sizes, train-test division, random seed, neural architecture, activation function, optimizer, and relevant stopping limits. These settings are not incidental details because changing them can alter the resulting predictions. [12,18,30,31]

A final thesis should distinguish clearly between analytically derived results, conventional numerical results, and machine-learning outputs obtained from actual computational runs. Numerical values should not be introduced merely to make a table appear complete. Where an experiment has not been performed, the appropriate mathematical framework can be stated without fabricating a result.

The same principle applies to convergence. An accurate approximation does not automatically establish formal convergence of an optimizer. The ODE experiment therefore reports both the achieved error and the fact that the specified function-evaluation limit was reached.

Limitations of the Experimental Study

- The selected benchmark problems are relatively simple and are intended to provide controlled verification rather than represent the full complexity of scientific computing.
- The cubic root experiment is restricted to a specified coefficient domain and to cases with a unique real root.
- The neural root model is an approximation and its largest observed error is substantially greater than its mean error.
- The ODE experiment concerns a smooth, non-stiff, one-dimensional equation over a short interval.
- The Euler benchmark with $h = 0.1$ does not represent the best accuracy achievable by conventional ODE solvers.
- The sine experiment demonstrates interpolation within $[0, 2\pi]$ and does not establish reliable extrapolation beyond the training domain.
- Neural-network results depend on architecture, initialization, optimizer, data split, and stopping conditions.
- Computational-time conclusions require hardware-controlled timing experiments if precise speed claims are to be made.
- The optimization section establishes the surrogate framework but does not insert an unverified machine-learning optimum.

Integrated Interpretation

Taken together, the experiments support a balanced interpretation of machine learning in mathematics. Machine learning can represent nonlinear input-output relationships and can incorporate mathematical constraints, but it does not remove the need for mathematical formulation, numerical validation, or error analysis.

The conventional and learning-based approaches should therefore be regarded as complementary. Mathematical methods provide exact relationships, convergence theory, constraints, and reliable reference solutions. Machine learning provides flexible approximation and rapid post-training evaluation. Their integration is most useful when the strengths of each approach are matched to the computational structure of the problem.

The findings also show why claims of universal superiority should be avoided. The correct comparison is problem dependent. For a single simple equation, a classical solver may be clearly preferable. For thousands of related equations, a trained mapping may become useful. For a differential equation, a mathematically constrained neural formulation may provide an alternative representation. For a known elementary function, the neural approximation is primarily a validation benchmark rather than a practical replacement.

Graphical Representation and Comparative Analysis

The following figures provide a visual representation of convergence, solution accuracy, and comparative error behaviour. They complement the numerical tables already reported in this paper and make the differences among the conventional and machine-learning approaches easier to interpret.

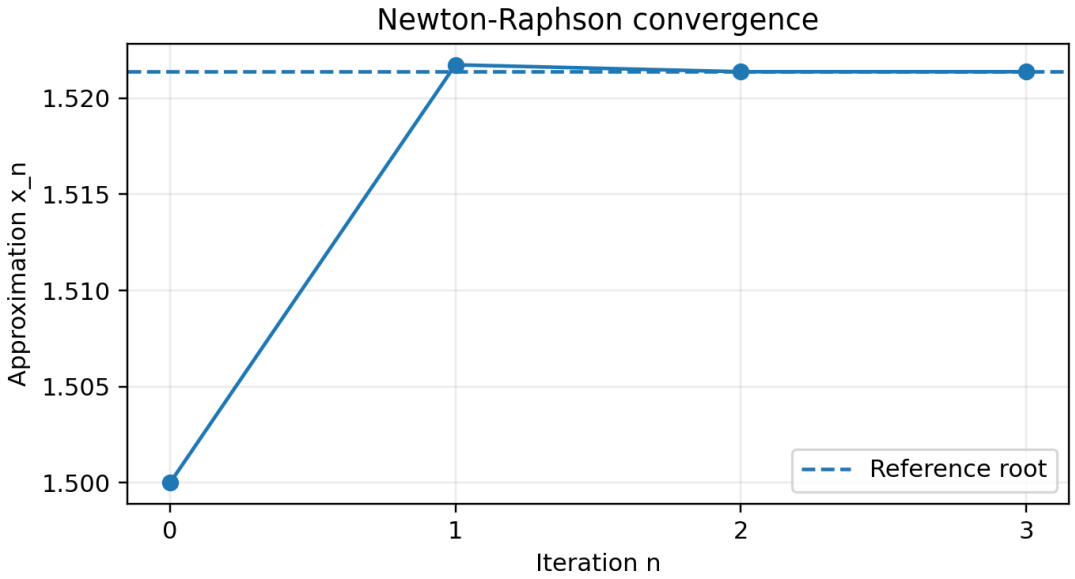

Figure 1.1 : Newton-Raphson convergence for the nonlinear algebraic equation.

The convergence curve shows that the approximation moves rapidly toward the reference root. The change between successive iterations becomes extremely small by the third iteration.

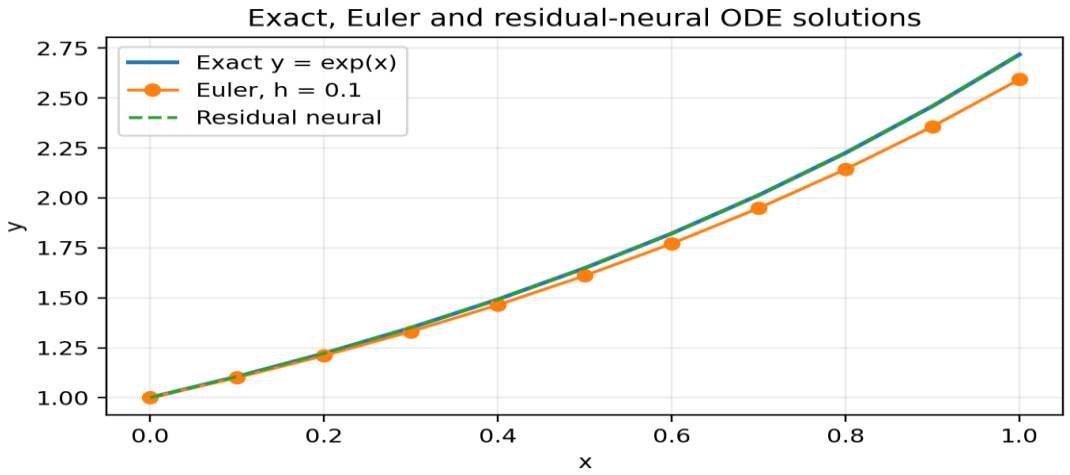

Figure 1.2: Comparison of the exact, Euler and residual-based neural solutions of the ODE.

The Euler curve departs progressively from the exact solution when h = 0.1, whereas the neural approximation visually overlaps the exact curve over the investigated interval.

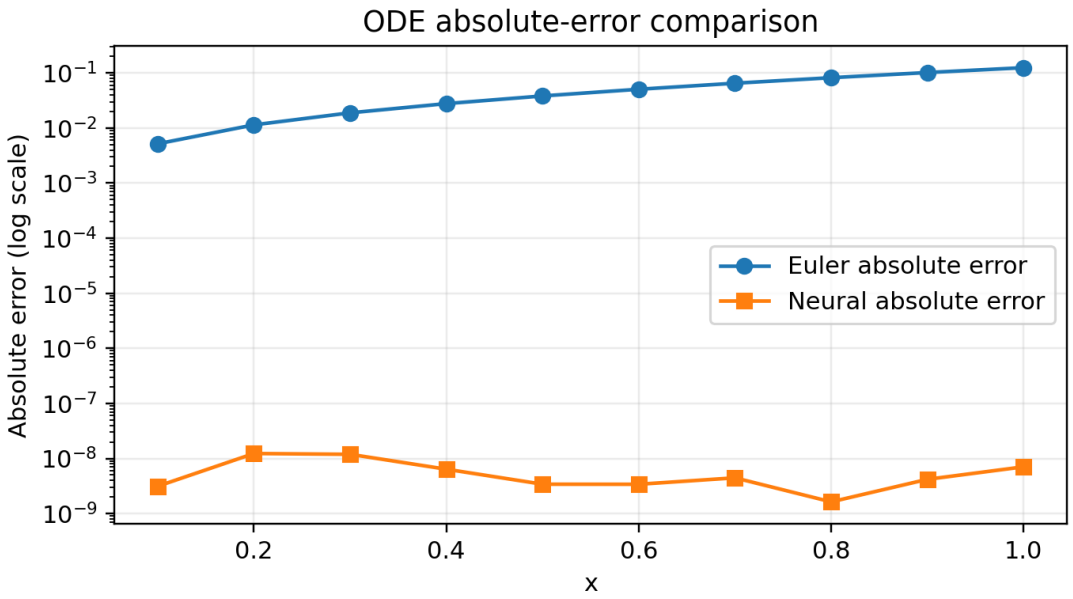

Figure 1.3 : Absolute-error comparison for Euler and neural ODE approximations.

The logarithmic error scale highlights the large difference between the coarse Euler error and the residual-based neural approximation for this particular elementary benchmark.

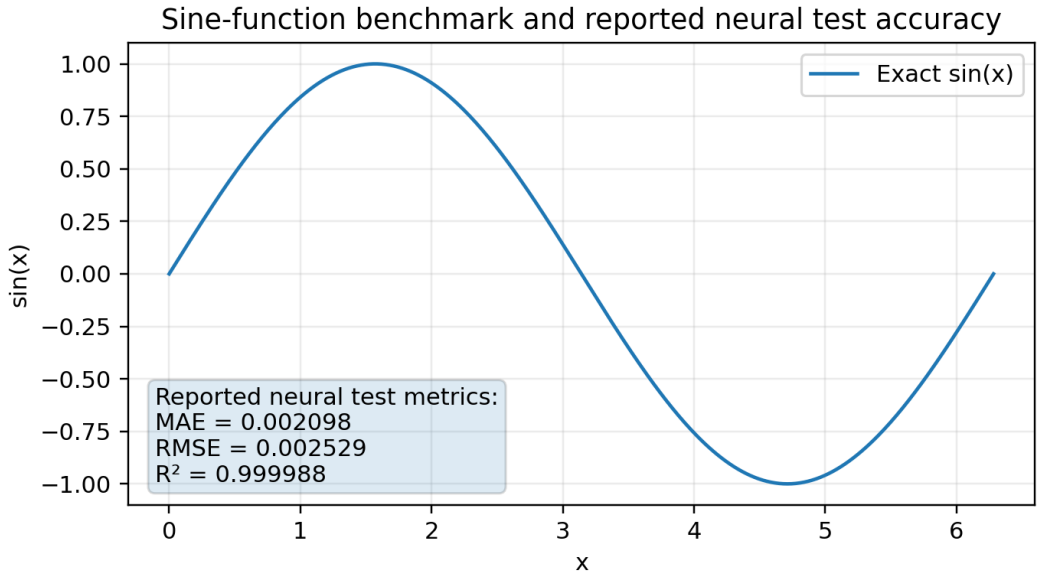

Figure 1.4 : Exact sine-function benchmark with the reported aggregate neural-network test metrics over [0, 2π].

The exact sine curve defines the benchmark used in the supervised experiment. Because the retained paper provides aggregate test metrics rather than the full pointwise neural prediction series, the figure reports MAE, RMSE and R^2 without fabricating a neural curve. The result concerns interpolation within the sampled interval.

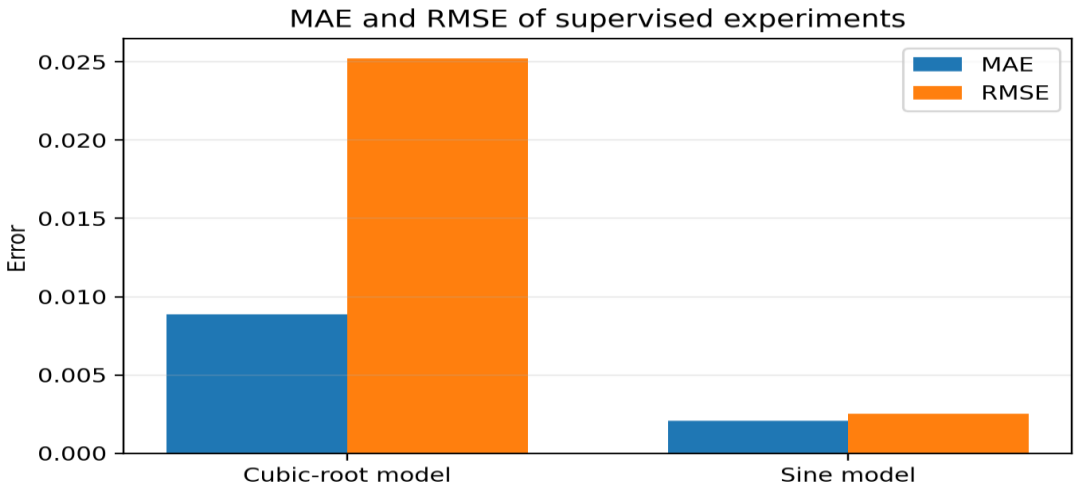

Figure 1.5: Comparison of MAE and RMSE for the supervised cubic-root and sine-function experiments.

The graphical comparison shows that both supervised experiments achieved small aggregate test errors, with the sine-function approximation producing the lower MAE and RMSE under the reported configurations.

Graphical Comparison Summary

Figure	Comparison represented	Main visual observation	Interpretation
Figure 1.1	Newton-Raphson iterations	Rapid stabilization near the root	Strong conventional convergence for the selected equation
Figure 1.2	Exact vs Euler vs Neural ODE	Neural curve overlaps exact; Euler deviates	Result is specific to the stated benchmark and settings
Figure 1.3	ODE absolute errors	Neural error is many orders smaller	Accuracy comparison should include solver order and training cost
Figure 1.4	Exact vs Neural sin(x)	Close agreement inside sampled domain	Supports interpolation, not unrestricted extrapolation
Figure 1.5	MAE and RMSE	Both supervised models show small test error	Multiple error measures provide a more complete assessment

Overall Interpretation of the Graphical Results

The graphical results reinforce the numerical findings without changing their scope. Newton-Raphson remains highly effective for the selected isolated nonlinear equation, while the parameterized learning experiment addresses a different repeated-prediction task.

For the ODE benchmark, the figures make the discretization error of Euler's method with h = 0.1 visually clear and show the close fit achieved by the residual-based neural model. This is a benchmark-specific comparison rather than a general ranking of neural and classical ODE solvers.

The sine-function graph demonstrates that the trained model follows the known nonlinear relationship closely within the investigated interval. The associated MAE and RMSE chart further emphasizes why graphical evidence should be interpreted together with quantitative error measures.Taken together, the figures improve the transparency of the experimental paper by allowing the reader to inspect convergence, curve agreement, and error magnitude directly rather than relying only on numerical tables.

Conclusion

This integrated paper presented the mathematical formulations, computational procedures, experimental results, error analysis, and interpretation of the principal problems investigated in the study. The nonlinear equation $x^3 - x - 2 = 0$ was solved using Newton–Raphson iteration, producing the approximate root $x = 1.5213797068$. The problem was then generalized to a parameterized cubic family, and a supervised neural model trained on 2,000 equations achieved a test MAE of 0.008862, RMSE of 0.025214, and R^2 of 0.999538.

The ordinary differential equation $dy/dx = y$ with $y(0) = 1$ was solved analytically as $y = e^x$ and numerically using Euler's method. With $h = 0.1$, the Euler approximation at $x = 1$ had an absolute error of approximately 0.124540. A residual-based neural trial solution incorporated the initial condition and differential equation directly and produced a very small reported approximation error on the selected benchmark, while the optimizer reached the stated function-evaluation limit.

The supervised sine-function experiment used 401 observations and produced a test MAE of 0.002098, RMSE of 0.002529, and R^2 of 0.999988. The optimization benchmark established the mathematical framework for surrogate modelling while retaining the exact solution $x^* = 3$ and $f(x^*) = 2$ as the reference.

The overall evidence indicates that machine learning can provide accurate and reusable approximations for selected mathematical problems, but its value depends on the structure of the task, the availability of conventional methods, the required numerical precision, the cost of training, and the reliability of predictions. The results support the use of machine learning as a complementary computational methodology integrated with mathematical reasoning rather than as a universal replacement for established mathematical techniques.

REFRENCES

[1] Burden, R. L., & Faires, J. D. (2011). Numerical Analysis (9th ed.). Cengage Learning. Used for standard numerical-analysis concepts including root-finding and initial-value methods.

[2] Lagaris, I. E., Likas, A., & Fotiadis, D. I. (1998). Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks, 9(5), 987–1000. doi:10.1109/72.712178.

[3] Raissi, M., Perdikaris, P., & Karniadakis, G. E. (2019). Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 378, 686–707. doi:10.1016/j.jcp.2018.10.045.

[4] Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. Neural Networks, 2(5), 359–366. doi:10.1016/0893-6080(89)90020-8.

[5] Hornik, K., Stinchcombe, M., & White, H. (1990). Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. Neural Networks, 3(5), 551–560. doi:10.1016/0893-6080(90)90005-6.

[6] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[7] Lu, L., Meng, X., Mao, Z., & Karniadakis, G. E. (2021). DeepXDE: A deep learning library for solving differential equations. SIAM Review, 63(1), 208–228. doi:10.1137/19M1274067.

[8] Mishra, S., & Molinaro, R. (2023). Estimates on the generalization error of physics-informed neural networks for approximating PDEs. IMA Journal of Numerical Analysis, 43(1), 1–43. doi:10.1093/imanum/drab093.

[9] Wang, S., Teng, Y., & Perdikaris, P. (2021). Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM Journal on Scientific Computing, 43(5), A3055–A3081. doi:10.1137/20M1318043.

[10] Krishnapriyan, A., Gholami, A., Zhe, S., Kirby, R., & Mahoney, M. W. (2021). Characterizing possible failure modes in physics-informed neural networks. Advances in Neural Information Processing Systems, 34, 26548–26560.

[11] Cuomo, S., Di Cola, V. S., Giampaolo, F., Rozza, G., Raissi, M., & Piccialli, F. (2022). Scientific machine learning through physics-informed neural networks: Where we are and what's next. Journal of Scientific Computing, 92, 88. doi:10.1007/s10915-022-01939-z.

[12] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[13] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems, 2, 303–314. doi:10.1007/BF02551274.

[14] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521, 436–444. doi:10.1038/nature14539.

[15] Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, 249–256.

[16] Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. Proceedings of the 3rd International Conference on Learning Representations.

[17] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic differentiation in machine learning: A survey. Journal of Machine Learning Research, 18(153), 1–43.

[18] Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

[19] Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning (2nd ed.). Springer. doi:10.1007/978-0-387-84858-7.

[20] Atkinson, K. E. (1989). An Introduction to Numerical Analysis (2nd ed.). Wiley.

[21] Chapra, S. C., & Canale, R. P. (2015). Numerical Methods for Engineers (7th ed.). McGraw-Hill Education.

[22] Liu, D. C., & Nocedal, J. (1989). On the limited memory BFGS method for large scale optimization. Mathematical Programming, 45, 503–528. doi:10.1007/BF01589116.

[23] Jones, D. R., Schonlau, M., & Welch, W. J. (1998). Efficient global optimization of expensive black-box functions. Journal of Global Optimization, 13, 455–492. doi:10.1023/A:1008306431147.

[24] Forrester, A. I. J., Sobester, A., & Keane, A. J. (2008). Engineering Design via Surrogate Modelling: A Practical Guide. Wiley. doi:10.1002/9780470770801.

[25] Queipo, N. V., et al. (2005). Surrogate-based analysis and optimization. Progress in Aerospace Sciences, 41(1), 1–28. doi:10.1016/j.paerosci.2005.02.001.

[26] Jin, Y. (2011). Surrogate-assisted evolutionary computation: Recent advances and future challenges. Swarm and Evolutionary Computation, 1(2), 61–70. doi:10.1016/j.swevo.2011.05.001.

[27] Nocedal, J., & Wright, S. J. (2006). Numerical Optimization (2nd ed.). Springer. doi:10.1007/978-0-387-40065-5.

[28] Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. Nature, 323, 533–536. doi:10.1038/323533a0.

[29] Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. Journal of the Society for Industrial and Applied Mathematics, 11(2), 431–441. doi:10.1137/0111030.

[30] Virtanen, P., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods, 17, 261–272. doi:10.1038/s41592-019-0686-2.

[31] Harris, C. R., et al. (2020). Array programming with NumPy. Nature, 585, 357–362. doi:10.1038/s41586-020-2649-2.