PHYSICS-INFORMED NEURAL NETWORKS AND CLASSICAL METHODS FOR SOLVING ORDINARY AND PARTIAL DIFFERENTIAL EQUATIONS

HONOUR'S PROJECT REPORT

Samir Karam
Department of Mathematics and Statistics
Concordia University
Montreal, Quebec
Supervised by
Dr. Simone Brugiapaglia
and Dr. Weiqi Wang

August 10, 2022

ABSTRACT

We present a brief overview of some of the contemporary methods of numerically solving boundary value problems in one and two dimensional domains. We first introduce the classical finite-difference method (FDM) for linear and nonlinear ODEs and the Ritz-Galerkin method with piecewise-linear and spline bases. Finally, we discuss a method of solving differential equations with neural networks and present examples that illustrate the success of each method. We finish with two examples that compare the efficacy of each method when we vary the degrees of freedom and remark on the results obtained.

1 Introduction

Deep learning has emerged as a new computational field in scientific machine learning that undercuts the traditional scientific method of theory and experimentation by relying heavily on data alone. Data-driven methods based on deep learning have revolutionized a diverse range of problems in statistics and applied mathematics such as image and speech recognition, classification, and regression [6] where large amounts of labeled data (consisting of inputs and correct outputs) are available. Where it is not, however, such as in the context of searching for specific solutions of partial differential equations, in which training data is typically unlabeled, it isn't clear how supervised learning methods can minimize a suitable loss function. However, new methods, such as those proposed by Lagaris et al. [5] and later coined physics-informed neural networks (PINN) by Raissi et al. [9], which uses unlabeled data from a set of grid points, has enabled deep learning to be used a function approximation system for emulating the solutions to various PDE [9][8]. In particular, the defining element of a PINN is the inclusion of a residual loss function that is based off of the underlying physical equation, which the network is trained to minimize [7], and that takes as input the unlabeled grid point data. This approach is remarkable in that no labeled data at all is required for the training process of the model to converge. The objective of this work is to demonstrate a rudimentary form of PINN, first proposed by Lagaris et al. [5], and compare our findings with some of the results from classical numerical analysis. To that effect, sections 2.1 and 2.2 introduce the classical methods of FDM and Ritz-Galerkin and section 2.3 provides a brief overview of deep learning as it is presented in Higham [3] and the method of Lagaris et al. [5]. As an aside, we also study experimentally obtained

rates of convergence of error as a function of step size in sections 3.1 and 3.2. We end with comparisons between the classical and deep learning method in section 3.7.

2 Theory

Here, we present the various methods of solving PDE that will be considered in this paper. We study finite-difference methods in section 2.1 for linear and nonlinear ODE as well for the Poisson equation in a rectangular domain. Ritz-Galerkin method for solving second-order BVP is discussed in section 2.2. In section 2.3, we present an overview of deep learning followed by the method of Lagaris et al. [5].

2.1 Finite-Difference methods

The following subsections present a way to solve second-order boundary-value problems by replacing the derivatives in the problem by a difference-quotient approximation. As we will see, the prototypical approach is to the divide an interval $[a, b]$ into equal subintervals delimited by an ordered sequence of points $a = x_0, x_1, \ldots, x_N = b$, and consider a discretized version of the ODE on these points, called grid points. Using this setup, let us now consider an exact solution to a differential equation, $u \in C^{N+1}[a, b]$ and some number $\xi(x)$, depending on x and lying between x_0 and x_N. Then, there is an approximation for u, relying on the Lagrange interpolating polynomial (see [1, p.108]):

$$u(x) = \sum_{k=0}^{N} u(x_k) \prod_{\substack{i=0 \\ i \neq k}}^{N} \frac{(x - x_i)}{(x_k - x_i)} + \frac{u^{(N+1)}(\xi(x))}{(N+1)!} \prod_{i=0}^{N} (x - x_i)$$

Taking a derivative with respect to x, we obtain

$$u'(x) = \sum_{k=0}^{N} u(x_k) \left(\prod_{\substack{i=0 \\ i \neq k}}^{N} \frac{(x - x_i)}{(x_k - x_i)} \right)' + \frac{u^{(N+1)}(\xi(x))}{(N+1)!} \sum_{k=0}^{N} \prod_{\substack{i=0 \\ i \neq k}}^{N} (x - x_i) + \left(\frac{u^{(N+1)}(\xi(x))}{(N+1)!} \right)' \prod_{i=0}^{N} (x - x_i)$$

Now, the goal is to derive an approximation for $u'(x)$ at the grid points, so we can take $x = x_j$ for some number $j \in \{0, \ldots, N\}$ and observe that all but one of the products in the middle term as well as the very last term vanish. We are left with:

$$u'(x_j) = \sum_{k=0}^{N} u(x_k) L'_k(x_j) + \frac{u^{(N+1)}(\xi(x_j))}{(N+1)!} \prod_{\substack{i=0 \\ i \neq j}}^{N} (x_j - x_i)$$

In this paper, we will only approximate $u'(x_j)$ using the nearest neighbour points of x_j, so we need only concern ourselves with the case $N = 2$. Moreover, if $h = (b-a)/N$ denotes the subinterval spacing, we can express differences of grid points in terms of h. For instance, $x_2 = x_0 + 2h$, $x_1 = x_0 + h$, and so on. These observations permit us to write the following simple expression, called the three-point midpoint formula for $u'(x)$:

$$u'(x_0) = \frac{u(x_0 + h) - u(x_0 - h)}{2h} - \frac{h^2}{6} u^{(3)}(\xi(x_0)) \tag{1}$$

where $\xi(x_0)$ is some number in $(x_0 - h, x_0 + h)$. Using equation (1) and an equation for the second derivative of u at x_0 will enable us to compute approximations for the class of second-order linear differential equations that are introduced in the next section.

2.1.1 Second-order linear ODE

Consider the second-order linear boundary-value problem:

$$u''(x) = p(x)u'(x) + q(x)u(x) + r(x), \quad \text{for } a \leq x \leq b, \text{ with } u(a) = \alpha \text{ and } u(b) = \beta \tag{2}$$

where $p, q,$ and r are continuous and $q(x) \geq 0$ on $[a, b]$. To solve this, we first divide the interval $[a, b]$ into $N + 1$ equal subintervals whose endpoints are the grid points $x_i = a + ih$, for $i \in \{0, \ldots, N + 1\}$, and $h = (b - a)/(N + 1)$ is the subinterval spacing. With this construction, we can rewrite equation (2) in a discrete version and for which the solution will be a vector containing the values of the solution at the grid points. Using this discretized version of our ODE,

$$u''(x_i) = p(x_i)u'(x_i) + q(x_i)u(x_i) + r(x_i), \quad \text{for } i = 1, 2, \ldots, N \tag{3}$$

we can use equation (1) in combination with a suitable approximation for the second-order derivative to obtain a linear system that, once solved, will approximate the exact solution at the grid points. To approximate the second-order derivative of u, we write the Taylor expansion for u about x_i:

$$u(x) = u(x_i) + u'(x_i)(x - x_i) + \frac{u''(x_i)}{2}(x - x_i)^2 + \frac{u'''(x_i)}{6}(x - x_i)^3 + \frac{u^{(4)}(\xi)}{24}(x - x_i)^4 \tag{4}$$

Evaluating (4) at the points $x_i + h$ and $x_i - h$, we get the following two equations:

$$u(x_i + h) = u(x_i) + u'(x_i)h + \frac{u''(x_i)}{2}h^2 + \frac{u'''(x_i)}{6}h^3 + \frac{u^{(4)}(\eta_i)}{24}h^4 \tag{5}$$

$$u(x_i - h) = u(x_i) - u'(x_i)h + \frac{u''(x_i)}{2}h^2 - \frac{u'''(x_i)}{6}h^3 + \frac{u^{(4)}(\zeta_i)}{24}h^4 \tag{6}$$

for some numbers $\eta_i \in (x_i, x_{i+1})$ and $\zeta_i \in (x_{i-1}, x_i)$. Adding (5) and (6) and using the Intermediate Value Theorem to simplify the error term yields an expression for the second derivative:

$$u''(x_i) = \frac{1}{h^2}\left(u(x_{i-1}) + u(x_{i+1}) - 2u(x_i)\right) - \frac{h^2}{12}u^{(4)}(\xi_i), \quad i = 1, 2, \ldots, N \tag{7}$$

for some number ξ_i in (x_{i-1}, x_{i+1}). Replacing x_0 by x_i in equation (1) allows us to substitute along with (7) into equation (3), which gives the equation

$$\frac{u(x_{i+1}) - 2u(x_i) + u(x_{i-1})}{h^2} = p(x_i)\left(\frac{u(x_{i+1}) - u(x_{i-1})}{2h}\right) + q(x_i)u(x_i) + r(x_i) + O(h^2)$$

for each $i = 1, 2, \ldots, N$. Letting $w_0 = u(a) = \alpha$, $w_{N+1} = u(b) = \beta$, and defining unknown values $w_i = u(x_i)$ for $1 \leq i \leq N$, we can rewrite the above equation in terms of the approximations w_i, with truncation error $O(h^2)$ (omitted below) and rearrange terms to get the following linear system:

$$-\left(\frac{h}{2}p(x_i) + 1\right)w_{i-1} + \left(h^2 q(x_i) + 2\right)w_i + \left(\frac{h}{2}p(x_i) - 1\right)w_{i+1} = -h^2 r(x_i), \quad i = 1, 2, \ldots, N \tag{8}$$

In matrix notation, equation (8) can be written as $Aw = b$, where w is the numerical approximation to u at the grid points x_i and is guaranteed to have a unique solution so long as h is chosen to be adequately small [1]. Important to note is that we required $u \in C^4[a, b]$ in the Taylor expansion of u about x_i, which permitted specifying the truncation error as $O(h^2)$. In section 3.1, we present several examples to highlight the convergence of this error as h tends to zero.

2.1.2 Second-order nonlinear ODE

For the general case, the system represented in (8) will not be linear and we must utilize an iterative approach called Newton's Method to calculate the approximations w_i. The general BVP is

$$u''(x) = f(x, u(x), u'(x)), \quad \text{for } a \leq x \leq b, \text{ with } u(a) = \alpha, \text{ and } u(b) = \beta \tag{9}$$

We assume firstly that f and its partial derivatives with respect to u and u' are continuous on the set

$$D = \{(x, u, u') \mid a \leq x \leq b, \text{ with } -\infty < u < \infty \text{ and } -\infty < u' < \infty\}$$

Moreover, we suppose that $f_u(x,u,u') \geq \delta$, for some $\delta > 0$ and that $f_u(x,u,u')$ and $f_{u'}(x,u,u')$ are bounded on D. These conditions are sufficient to guarantee the existence of a unique solution [1, Thm 11.1]. As in the linear case, we divide the interval $[a,b]$ into equal subintervals delimited by the grid points $x_i = a + ih$ for $i = 0, 1, \ldots, N+1$ and assuming that the exact solution has $u \in C^4[a,b]$, we can use the centred difference equations (1) and (7) with $O(h^2)$ truncation error in

$$u''(x_i) = f(x_i, u(x_i), u'(x_i)) \tag{10}$$

to give the equation

$$\frac{u(x_{i-1}) - 2u(x_i) + u(x_{i+1})}{h^2} = f\left(x_i, u(x_i), \frac{u(x_{i+1}) - u(x_{i-1})}{2h} - \frac{h^2}{6}u'''(\eta_i)\right) + \frac{h^2}{12}u^{(4)}(\xi_i)$$

Again, letting $w_0 = u(a) = \alpha$, $w_{N+1} = u(b) = \beta$, and $w_i = u(x_i)$ for $1 \leq i \leq N$ and omitting the error terms we get the following nonlinear system

$$w_{i-1} - 2w_i + w_{i+1} - h^2 f\left(x_i, w_i, \frac{w_{i+1} - w_{i-1}}{2h}\right) = 0, \quad \text{for each } i = 1, 2, \ldots, N \tag{11}$$

which has a solution for sufficiently small h [1, p.707]. To solve this system, we use Newton's method, which involves fixing a starting approximation $\left\{ \left(w_1^{(0)}, w_2^{(0)}, \ldots, w_N^{(0)}\right)^t \right\}$, and producing a sequence of iterates $\left\{ \left(w_1^{(k)}, w_2^{(k)}, \ldots, w_N^{(k)}\right)^t \right\}$ using the recursive formula

$$w_i^{(k)} = w_i^{(k-1)} + v_i, \quad \text{for each } i = 1, 2, \ldots, N \tag{12}$$

Here, the v_i are the components of the solution to the linear matrix equation $Jv = A$, where A is the vector whose i^{th} component equals the left hand side of equation (11) and J is the Jacobian matrix of derivatives of (11) with respect to w_i, for each $i = 1, 2, \ldots, N$.

As in the linear case, the truncation error at the grid points for this method is $O(h^2)$. Section 3.2 highlights this with several examples.

2.1.3 Finite-Difference method for the Poisson equation in a rectangular domain

The method provided here and taken from Burden et al. [1, ch.12.1] is an extension of the finite-difference method of section 2.1.1 to the case of two dimensional elliptic PDE. Consider the following boundary value problem:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x,y) \tag{13}$$

where $u = u(x,y)$ and x and y are taken over the space of all points in a square 2-dimensional region R delimited by $a, b, c,$ and d. That is, $R = \{(x,y) \mid a < x < b,\ c < y < d\}$. Let S denote the boundary of R and suppose that, for points appearing on S, we have continuous Dirichlet boundary conditions such that $u(a,y) = f_0(y)$, $u(b,y) = f_1(y)$ and $u(x,c) = g_0(x)$, $u(x,d) = g_1(x)$. To solve this problem using finite-differences, we discretize R by a uniform grid of rectangles and rewrite equation (13) in a discrete form over the grid points forming the vertices of each rectangle. Let N and M denote the number of subintervals on each axis. Then, we have the subinterval spacings h and k given by $h = (b-a)/N$ and $k = (d-c)/M$. We can now define the coordinates of each grid point by $x_i = a + ih$ and $y_j = c + jk$ for $i = 0, \ldots, N$ and $j = 0, \ldots, M$. Finally, using this discretized version of R, we rewrite (13) as:

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) + \frac{\partial^2 u}{\partial y^2}(x_i, y_j) = f(x_i, y_j), \quad \text{for each } i = 0, 1, \ldots, N, \text{ and } j = 0, 1, \ldots, M \tag{14}$$

Since, the grid points found on the boundary of R are already exactly determined by the function g, it is not necessary to look for approximations there and we need only consider centred-difference formulas for the points that lie in the

interior, that is, when $i = 1, \ldots, N-1$ and $j = 1, \ldots, M-1$. Holding y fixed, we can rewrite equation (7) in the 2D case to approximate the second partial derivative of u with respect to x:

$$\frac{\partial^2 u}{\partial x^2}(x_i, y_j) = \frac{u(x_{i-1}, y_j) - 2u(x_i, y_j) + u(x_{i+1}, y_j)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i, y_j) \tag{15}$$

for some $\xi_i \in (x_{i-1}, x_{i+1})$. Similarly, holding x fixed, we have:

$$\frac{\partial^2 u}{\partial y^2}(x_i, y_j) = \frac{u(x_i, y_{j-1}) - 2u(x_i, y_j) + u(x_i, y_{j+1})}{k^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j) \tag{16}$$

for some $\eta_j \in (y_{j-1}, y_{j+1})$. Substituting the above two equations into (14), we obtain

$$\frac{u(x_{i-1}, y_j) - 2u(x_i, y_j) + u(x_{i+1}, y_j)}{h^2} + \frac{u(x_i, y_{j-1}) - 2u(x_i, y_j) + u(x_i, y_{j+1})}{k^2} = f(x_i, y_j) + \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i, y_j) + \frac{h^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i, \eta_j)$$

for each $i = 1, \ldots, N-1$ and $j = 1, \ldots, M-1$. For points on the boundary, we evaluate u according to the conditions given by the functions $f_0(y)$, $f_1(y)$, $g_0(x)$, and $g_1(x)$. To simplify the notation, we ignore the truncation error and introduce the unknowns $w_{ij} = u(x_i, y_j)$, for each $i = 1, \ldots, N-1$ and $j = 1, \ldots, M-1$. Rearranging terms, we get the following $(N-1)(M-1) \times (N-1)(M-1)$ linear system

$$2\left[\left(\frac{h}{k}\right)^2 + 1\right] w_{ij} - (w_{i-1,j} + w_{i+1,j}) - \left(\frac{h}{k}\right)^2 (w_{i,j-1} + w_{i,j+1}) = -h^2 f(x_i, y_j)$$

with truncation error $O(h^2 + k^2)$. We give 2 examples to illustrate the feasibility of this method in section 3.3, in which we have chosen to use a square domain with $N = M$ and $h = k$ to keep the problem simple.

2.2 Ritz-Galerkin method

The next numerical method of interest is the Ritz-Galerkin approach, which restructures the boundary-value problem as a variational one. Consider the following second-order BVP:

$$-\frac{d}{dx}\left(p(x)\frac{du}{dx}\right) + q(x)u = f(x), \quad \text{for } 0 \le x \le 1, \quad u(0) = u(1) = 0 \tag{17}$$

Suppose p is positive and continuous on $[0, 1]$. Also, suppose that q and f are continuous functions on $[0, 1]$. Then, the conclusion of the following theorem from Burden et al. [1, Thm 11.4] holds:

Theorem 1. *Let $p \in C^1[0, 1]$, q, f be continuous functions on $[0, 1]$, and $p(x) \ge \delta > 0$ and $q(x) \ge 0$ for $x \in [0, 1]$. Then, a function $u \in C^2[0, 1]$ is the unique solution to BVP* (17) *if and only if u minimizes the integral*

$$I[y(x)] = \int_0^1 \{p(x)[y'(x)]^2 + q(x)[y(x)]^2 - 2f(x)y(x)\}dx \tag{18}$$

Now, the goal is to approximate the solution u by minimizing integral (18) over linear combinations of certain sets of basis functions $\phi_1, \phi_2, \ldots \phi_N$ in $C^2[0, 1]$ that also satisfy the boundary conditions in (17) and are linearly independent. Writing the solution u as a linear combination of these basis functions with coefficients c_i for each $i = 1, 2, \ldots, N$ and substituting into (17) gives

$$I\left[\sum_{i=1}^N c_i\phi_i\right] = \int_0^1 \left\{p(x)\left[\sum_{i=1}^N c_i\phi_i'(x)\right]^2 + q(x)\left[\sum_{i=1}^N c_i\phi_i(x)\right]^2 - 2f(x)\sum_{i=1}^N c_i\phi_i(x)\right\}dx$$

Taking a derivative with respect to the j^{th} coefficient.

$$\frac{\partial I}{\partial c_j} = \int_0^1 \left\{ 2p(x) \left[\sum_{i=1}^{N} c_i \phi_i'(x) \phi_j'(x) \right] + 2q(x) \left[\sum_{i=1}^{N} c_i \phi_i(x) \phi_j(x) \right] - 2f(x)\phi_j(x) \right\} dx$$

A necessary (yet insufficient) condition to ensure a minimum is that the partial derivatives with respect to c_j for each $j = 1, 2, \ldots, N$ equal zero.

$$\sum_{i=1}^{n} \left[\int_0^1 p(x)\phi_i'(x)\phi_j'(x) + q(x)\phi_i(x)\phi_j(x)dx \right] c_i - \int_0^1 f(x)\phi_j(x)dx = 0, \quad \text{for each } j = 1, 2, \ldots, N \tag{19}$$

Equation (19) is a linear system $Ac = b$ with $A = [a_{ij}]$ where $a_{ij} = \int_0^1 p(x)\phi_i'(x)\phi_j'(x) + q(x)\phi_i(x)\phi_j(x)dx$, and the components of b are given by $b_i = \int_0^1 f(x)\phi_i(x)dx$. This system is easily solved once a suitable basis has been fixed, with the exception being some of the integrals that are often more easily computed numerically. In this paper, we deal with examples where the integrals a_{ij} and b_i are easily computed symbolically and so this extra step is avoided.

2.2.1 The Piecewise-Linear Basis

We begin by choosing a partition of $[0, 1]$, delimited by a sequence of increasing points x_i to serve as grid points. We define the subinterval spacings $h_i = x_{i+1} - x_i$ for each $i = 0, 1, \ldots, N$ and present the "hat" functions

$$\phi_i(x) = \begin{cases} 0 & \text{if } 0 < x \leq x_{i-1} \\ \frac{1}{h_{i-1}}(x - x_{i-1}) & \text{if } x_{i-1} < x \leq x_i \\ \frac{1}{h_i}(x_{i+1} - x) & \text{if } x_i < x \leq x_{i+1} \\ 0 & \text{if } x_{i+1} < x \leq 1 \end{cases}$$

By construction, the approximation $\phi(x) = \sum_{i=1}^{N} c_i \phi_i(x)$ is a piecewise continuous interpolation of the discrete approximations of u at the grid points. In other words, unlike in the previous sections, this new method returns a function over the given interval, not just a vector of approximations at the grid points. To compute the integrals in (19), we must compute the derivative $\phi_i'(x)$

$$\phi_i'(x) = \begin{cases} 0 & \text{if } 0 < x < x_{i-1} \\ \frac{1}{h_{i-1}} & \text{if } x_{i-1} < x < x_i \\ -\frac{1}{h_i} & \text{if } x_i < x < x_{i+1} \\ 0 & \text{if } x_{i+1} < x < 1 \end{cases}$$

and using the fact that the products $\phi_i(x)\phi_j(x) \equiv 0$ and $\phi_i'(x)\phi_j'(x) \equiv 0$ unless $j = i$ or $j = i \pm 1$, we calculate explicit forms for the matrix elements a_{ij} and b_i. On the main diagonal, we have $j = i$ and $i = 1, 2, \ldots, N$

$$\begin{aligned} a_{ii} = &\left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} p(x)dx + \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} q(x)(x - x_{i-1})^2 dx + \\ &+ \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} p(x)dx + \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} q(x)(x_{i+1} - x)^2 dx \end{aligned}$$

On the superdiagonal, we have $j = i + 1$ and $i = 1, 2, \ldots, N - 1$

$$a_{i,i+1} = -\left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} p(x)dx + \left(\frac{1}{h_i}\right)^2 \int_{x_i}^{x_{i+1}} q(x)(x_{i+1} - x)(x - x_i)dx$$

On the subdiagonal, we have $j = i - 1$ and $i = 2, 3, \ldots, N$

$$a_{i,i-1} = -\left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} p(x)dx + \left(\frac{1}{h_{i-1}}\right)^2 \int_{x_{i-1}}^{x_i} q(x)(x - x_{i-1})(x_i - x)dx$$

and finally, we have the vector elements b_i, for $i = 1, 2, \ldots, N$

$$b_i = \frac{1}{h_{i-1}} \int_{x_{i-1}}^{x_i} f(x)(x - x_{x-1})dx + \frac{1}{h_i} \int_{x_i}^{x_{i+1}} f(x)(x_{i+1} - x)dx$$

Although we do not prove it, it can be shown that an upper bound for the error is $O(h^2)$ [1, p.720]. This result is highlighted in section 3.4 where the error is plotted sequentially for various values of the subinterval spacings h_i, which are taken to be equal between all grid points.

2.2.2 The Bell-Shaped Splines Basis

A secondary choice of basis makes use of interpolatory splines to form a continuous approximation ϕ. Consider five grid points $\{x_i\}_{i=0}^4$ and some function $f \in C^2[x_0, x_4]$. The standard interpolatory spline $S(x)$ is constructed to satisfy the following criteria:

1. $S(x)$ is a cubic polynomial, which we denote by $S_i(x)$ on the interval $[x_i, x_{i+1}]$ for $i = 0, 1, 2, 3$
2. $S_i(x_i) = f(x_i)$ and $S_i(x_{i+1}) = f(x_{i+1})$ for $i = 0, 1, 2, 3$
3. $S_{i+1}(x_{i+1}) = S_i(x_{i+1})$ for $i = 0, 1, 2$
4. $S'_{i+1}(x_{i+1}) = S'_i(x_{i+1})$ for $i = 0, 1, 2$
5. $S''_{i+1}(x_{i+1}) = S''_i(x_{i+1})$ for $i = 0, 1, 2$
6. and one of (boundary conditions)
   (a) $S''(x_0) = S''(x_N) = 0$
   (b) $S'(x_0) = f'(x_0)$ and $S'(x_N) = f'(x_N)$

Since 1 defines 16 constants (4 constants per polynomial, of which there are 4), we must normally choose only one of (a) or (b) (each providing 2 additional constraints) to have 16 constraints (8 originate from criterion 2 and 6 come from criteria 4 and 5) and to guarantee a unique solution. Note that the constraints given by 3 are already implied by 2. But this basis requires that both boundary constraints are satisfied, which forces a relaxation of two constraints. A workaround is to modify criterion 2 to only apply on the odd grid points (even values of i):

2. $S(x_i) = f(x_i)$ for $i = 0, 2, 4$

Now, there are 4 constraints from 2 and 2 constraints from 3 (which are no longer implied). Adding these to the 6 constraints from points 4 and 5 and the 4 constraints imposed by the boundary conditions gives 16 in all. Setting $x_0 = -2$, $x_1 = -1$, $x_2 = 0$, $x_3 = 1$, and $x_4 = 2$, we define the bell-shaped spline, satisfying the conditions $S(x_0) = 0$, $S(x_2) = 1$, and $S(x_4) = 0$ as well as both boundary conditions with $f'(x_0) = f'(x_n) = 0$.

$$S(x) = \begin{cases} 0, & \text{if } x \leq -2, \\ \frac{1}{4}(2+x)^3, & \text{if } -2 < x \leq -1, \\ \frac{1}{4}\left[(2+x)^3 - 4(1+x)^3\right], & \text{if } -1 < x \leq 0, \\ \frac{1}{4}\left[(2-x)^3 - 4(1-x)^3\right], & \text{if } 0 < x \leq 1, \\ \frac{1}{4}(2-x)^3, & \text{if } 1 < x \leq -2, \\ 0, & \text{if } 2 < x, \end{cases} \tag{20}$$

Now, partitioning $[0,1]$ into equally spaced subintervals of length $h = 1/(N+1)$ with the grid points $x_i = ih$ for each $i = 0, 1, \ldots, N+1$, we construct $N+2$ linearly independent basis functions $\{\phi_i\}_{i=0}^{N+1}$ based on the spline (20):

$$\phi_i(x) = \begin{cases} S\left(\frac{x}{h}\right) - 4S\left(\frac{x+h}{h}\right), & \text{if } i = 0, \\ S\left(\frac{x-h}{h}\right) - S\left(\frac{x+h}{h}\right), & \text{if } i = 1, \\ S\left(\frac{x-ih}{h}\right), & \text{if } 2 \leq i \leq N-1, \\ S\left(\frac{x-Nh}{h}\right) - S\left(\frac{x-(N+2)h}{h}\right), & \text{if } i = N, \\ S\left(\frac{x-(N+1)h}{h}\right) - 4S\left(\frac{x-(N+2)h}{h}\right), & \text{if } i = N+1, \end{cases} \tag{21}$$

And the matrix components a_{ij} and b_i are calculated as before, now with $i, j = 0, 1, \ldots, N+1$. Remarkably, the use of bell-shaped splines provides a significant improvement over the piecewise-linear basis of the previous section, with an upper error bound of $O(h^4)$. Although we do not prove this, we do provide an illustration of this fact in the results of section 3.4.

2.3 Physics-Informed Neural Networks

In this section, we present a completely different approach put forth by Lagaris et al. [5] and later coined *physics-informed* by Raissi et al. [9] in which a feedforward neural network is trained on a uniform grid to solve 1D and 2D differential equations. To tune the parameters of the network, we use a loss function that is based on the residual error incurred by substituting the model output into the differential equation. The model is then trained by updating the network parameters according to the gradient of the loss function at each step.

2.3.1 A quick introduction to Artificial Neural Networks

To understand the mechanics of ANNs, we introduce an example of a 4-layer feedforward network that takes as input a vector $\mathbf{x} \in \mathbb{R}^2$ and outputs a vector $\mathbf{N}(\mathbf{x}) \in \mathbb{R}^2$ that determines the "type" of point on a planar domain. Consider the square $[0,1] \times [0,1]$ containing the following points:

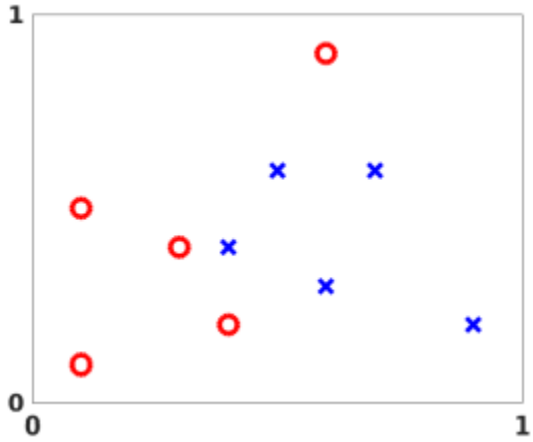


Figure 1: Red circles and blue X's in $\mathbb{R}^2$ (image taken from Higham [3])

Then, a *trained* neural network should be able to correctly classify any of the above points, called training data, as either red circles or blue X's as well as compute predictions for other points in the square. This can be done by defining some target function

$$\mathbf{y}(\mathbf{x}) = \begin{cases} [1 \;\; 0]^T & \text{if } \mathbf{x} \text{ is a red circle} \\ [0 \;\; 1]^T & \text{if } \mathbf{x} \text{ is a blue X} \end{cases}$$

(where T denotes the vector transpose) and a rule on the components of $\mathbf{N}(\mathbf{x})$, such as: $\mathbf{x} \in \mathbb{R}^2$ is a red circle if the first component is greater than (or equal to) the second component, and a blue X otherwise. Consider the 4-layer feedforward network of Figure 2:

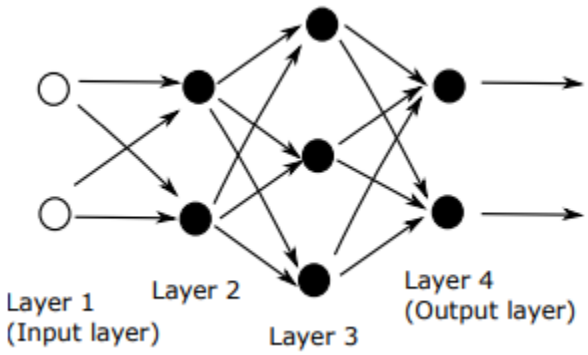


Figure 2: A network with four layers, which takes as input a vector $\mathbf{x} \in \mathbb{R}^2$ and produces as output a vector $\mathbf{N}(\mathbf{x}) \in \mathbb{R}^2$ (image taken from Higham [3])

In this construction, the hollow points, or input nodes are the components of the input vector $\mathbf{x}$ and the solid points are the "neurons" of the network. The latter output a real number whose value is based on a weighted sum, according to some matrix of *weights*, of each of the inputs in the previous layer plus its own implicit *bias*. Each layer acts by collectively producing a vectorized output with dimension equal to the number of nodes and in which each component of the output has been acted on by some function $\sigma : \mathbb{R} \to \mathbb{R}$, called an activation function. For example, layer 2, which is composed of 2 neurons, receives input in $\mathbb{R}^2$ and thus has the following weight matrix:

$$W^{(2)} = \begin{bmatrix} w_{1\to1} & w_{2\to1} \\ w_{1\to2} & w_{2\to2} \end{bmatrix}$$

If $\mathbf{x} = [x_1 \;\; x_2]^T$ is the output of the input layer and $\mathbf{b}^{(2)} = [b_1 \;\; b_2]^T$ is the vector of biases corresponding to the neurons in layer 2, the output of this layer is given by

$$\boldsymbol{\sigma}\left(W^{(2)}x + \mathbf{b}^{(2)}\right) \in \mathbb{R}^2$$

Likewise, the third layer is composed of three neurons that each receive input from the two neurons in the previous layer. Thus, the weight matrix for this layer can be represented by

$$W^{(3)} = \begin{bmatrix} w_{1\to1} & w_{2\to1} \\ w_{1\to2} & w_{2\to2} \\ w_{1\to3} & w_{2\to3} \end{bmatrix}$$

and the corresponding output at this layer is

$$\boldsymbol{\sigma}\left(W^{(3)}\boldsymbol{\sigma}\left(W^{(2)}x + \mathbf{b}^{(2)}\right) + \mathbf{b}^{(3)}\right) \in \mathbb{R}^3$$

where $\mathbf{b}^{(3)}$ is the vector of biases for the neurons in the fourth layer. Finally, the weight matrix of inputs to layer 4 is

$$W^{(4)} = \begin{bmatrix} w_{1\to1} & w_{2\to1} & w_{3\to1} \\ w_{1\to2} & w_{2\to2} & w_{3\to2} \end{bmatrix}$$

and we can now write the output of the whole network $\mathbf{N}(\mathbf{x})$ as

$$\boldsymbol{\sigma}\left(W^{(4)}\boldsymbol{\sigma}\left(W^{(3)}\boldsymbol{\sigma}\left(W^{(2)}x+b^{(2)}\right)+b^{(3)}\right)+\mathbf{b}^{(4)}\right)\in\mathbb{R}^2$$

where $\mathbf{b}^{(4)}$ is the vector of biases for the neurons in the fourth layer. Thus, we have shown that the function $\mathbf{N}:\ \mathbb{R}^2\to\mathbb{R}^2$ is composed of 23 parameters in all, of which 16 are weights and 7 are biases. The task becomes to ensure that these parameters are well tuned to approximate the correct outputs with respect to the training data. This step, called the training step, involves calculating the gradient of some *loss* function, and adjusting the parameters accordingly to eventually minimize it. This process is repeated iteratively until the network works well enough to be considered "trained", in the sense that it correctly classifies the labeled training data and hopefully also other data in the domain. We define a basic form for a loss function over N data points as

$$loss=\sum_{i=1}^{N}\|y(x_i)-\mathbf{N}(x_i)\|_2^2$$

where here we have used $\mathbf{x}_i$ to denote the i^{th} data point and $\mathbf{y}(x_i)$ and $\mathbf{N}(x_i)$ to denote, respectively, the i^{th} target and model outputs. The difficulty is now reduced to the computation of the gradient of the $loss$ function with respect to the network parameters. In this work, we facilitate this task by making use of the Keras module which has built-in algorithms for loss minimization. After several updates of the parameters, called epochs, we can expect the following visualization of the result, in which the grey zone contains points that are classified as red circles, and the white zone contains points that are classified as blue X's:

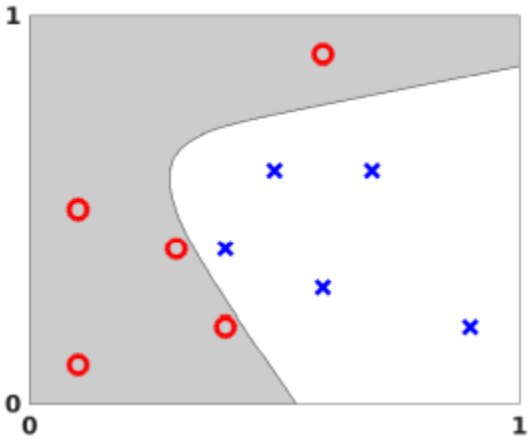


Figure 3: Visualisation of the output from the trained network (image taken from Higham [3])

2.3.2 A rudimentary approach to solving differential equations using PINN

Now, we are ready to introduce a simple method of solving differential equations that uses an artificial neural network combined with a residual loss function to approximate the solution function using unlabeled grid points [5]. Consider the following general description of a differential equation subject to some boundary conditions

$$K\left(\mathbf{x},u(\mathbf{x}),u'(\mathbf{x}),u''(\mathbf{x})\right)=0,\quad \mathbf{x}=(x_1,x_2,\ldots,x_n)\in R\subset\mathbb{R}^n \tag{22}$$

where $'$ and $''$ denote respectively the first and second derivatives of $u(x)$ with respect to one of the components of $\mathbf{x}$. By dividing the domain R and its boundary into countably many subdomains delimited by the points $\mathbf{x}_i$, we can write a discrete version of (22)

$$K\left(\mathbf{x}_i,u(\mathbf{x}_i),u'(\mathbf{x}_i),u''(\mathbf{x}_i)\right))=0,\quad \forall\mathbf{x}_i=(x_{i1},x_{i2},\ldots,x_{in})\in R\subset\mathbb{R}^n \tag{23}$$

Clearly, any solution to (22) must satisfy equation (23). We can write a basic form for a loss function by using the fact that (23) is required to equal zero for any solution that satisfies it.

$$\sum_{\mathbf{x}_i \in R} K\left(\mathbf{x}_i, u_t(\mathbf{x}_i, \mathbf{p}), Du_t(\mathbf{x}_i, \mathbf{p}), D^2 u_t(\mathbf{x}_i, \mathbf{p})\right)^2 \tag{24}$$

Where $u_t(\mathbf{x}, \mathbf{p})$ is a trial solution that is a function of the data point $\mathbf{x}_i$ and of some vector of adjustable parameters $\mathbf{p}$. Thus, the problem becomes to find a function $u_t(\mathbf{x}, \mathbf{p})$ that minimizes the loss function in (24) over the choice of parameters subject to the constraints imposed by the boundary conditions. This is a constrained optimization problem that happens to be very easily reduced to an unconstrained one by writing the trial function as a sum of two terms:

$$u_t(\mathbf{x}, \mathbf{p}) = A(\mathbf{x}) + B\left(\mathbf{x}, \mathbf{N}(\mathbf{x}, \mathbf{p})\right)$$

By construction, $A(\mathbf{x})$ is made to satisfy the boundary conditions, while $B\left(\mathbf{x}, \mathbf{N}(\mathbf{x}, \mathbf{p})\right)$ vanishes at the boundary and contains adjustable parameters that are embedded into a neural network $\mathbf{N}(\mathbf{x}, \mathbf{p})$. Thus, the problem of choosing $\mathbf{p}$ is reduced to an unconstrained minimization problem which can be solved by a suitable gradient descent algorithm [5].

2.3.3 The model

To approximate the trial solution that minimizes equation (24), we use a much more basic neural network structure (proposed by Lagaris et al. [5]) than the one employed in section 2.3.1. Here, we will consider a feedfoward neural network with n input units connected to a hidden layer (between the input and output layers) with H sigmoid [10] activation neurons, which is then fed into a single linear output neuron. If w_{ij} denotes the weight from the j^{th} input node to the i^{th} hidden neuron and b_i is the bias of the i^{th} hidden neuron, then $z_i = \sum_{j=1}^{n} w_{ij}x_j + b_i$ is the output of the i^{th} neuron in the hidden layer. The output of the whole network is given by

$$\mathbf{N} = \sum_{i=1}^{H} v_i \sigma(z_i) \tag{25}$$

where v_i is the weight on the input from each hidden neuron into the single linear output neuron. In the results that follow, we will be using the Adam optimizer from Keras [2][4] to automate the minimization, which involves computing gradients, but it is still useful to visualize how these calculations could be done in theory. Since the loss functions that we will be considering are functions of the trial solution and its derivatives with respect to the inputs (as in (24)), we will need to first compute the derivatives of $\mathbf{N}$ with respect to the inputs before computing the derivatives of $\mathbf{N}$ with respect to the model parameters. The latter is of primary importance since the goal is to adjust the parameters in each epoch according to the gradient of the loss function with respect to the model parameters. Substituting z_i for $\sum_{j=1}^{n} w_{ij}x_j + b_i$ in (25) and taking derivatives, we obtain

$$\begin{aligned}
\mathbf{N} &= \sum_{i=1}^{H} v_i \sigma\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right) \\
\frac{\partial \mathbf{N}}{\partial x_j} &= \sum_{i=1}^{H} v_i w_{ij} \sigma'\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right) \\
\frac{\partial^2 \mathbf{N}}{\partial x_j^2} &= \sum_{i=1}^{H} v_i w_{ij}^2 \sigma''\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right) \\
&\vdots \\
\frac{\partial^k \mathbf{N}}{\partial x_j^k} &= \sum_{i=1}^{H} v_i w_{ij}^k \sigma^{(k)}\left(\sum_{j=1}^{n} w_{ij}x_j + b_i\right)
\end{aligned}$$

Furthermore, in the case of needing to differentiate with respect to multiple components subsequently, we can produce the following:

$$\frac{\partial \mathbf{N}}{\partial x_1} = \sum_{i=1}^{H} v_i w_{i1} \sigma' \left(\sum_{j=1}^{n} w_{ij} x_j + b_i \right)$$
$$\frac{\partial^2 \mathbf{N}}{\partial x_1 x_2} = \sum_{i=1}^{H} v_i w_{i1} w_{i2} \sigma' \left(\sum_{j=1}^{n} w_{ij} x_j + b_i \right)$$

It is now easy to visualize the general case:

$$\frac{\partial^{\lambda_1}}{\partial x_1^{\lambda_1}} \frac{\partial^{\lambda_2}}{\partial x_2^{\lambda_2}} \cdots \frac{\partial^{\lambda_n}}{\partial x_n^{\lambda_n}} \mathbf{N} = \sum_{i=1}^{H} v_i \prod_{k=1}^{n} w_{ik}^{\lambda_k} \sigma \left(\sum_{j=1}^{n} w_{ij} x_j + b_i \right)^{(\Lambda)} = \sum_{i=1}^{H} v_i P_i \sigma_i^{(\Lambda)} \tag{26}$$

where $\Lambda = \sum_{j=1}^{n} \lambda_j$. Denoting the right hand side of equation (26) by $\mathbf{N}_x$, we can now easily differentiate with respect to the network parameters v_i, and b_i:

$$\frac{\partial \mathbf{N}_x}{\partial v_i} = P_i \sigma_i^{(\Lambda)} \tag{27}$$

$$\frac{\partial \mathbf{N}_x}{\partial b_i} = \sum_{i=1}^{H} v_i P_i \sigma_i^{(\Lambda+1)} \tag{28}$$

and using the product rule to differentiate with respect to w_{ij},

$$\frac{\partial \mathbf{N}_x}{\partial w_{ij}} = v_i \lambda_j w_{ij}^{\lambda_j - 1} \Big(\prod_{\substack{k=1 \\ k \neq j}} w_{ik}^{\lambda_k} \Big) \sigma_i^{(\Lambda)} + v_i P_i x_j \sigma_i^{(\Lambda+1)} \tag{29}$$

2.3.4 Breakdown of the trial solution for various cases

The approach presented in section 2.3.2 requires that the trial solution is written as the sum of two terms $A(x)$ and $B(x, \mathbf{N}(x, \mathbf{p}))$, where $A(x)$ is constructed to satisfy the boundary conditions and $B(x, N(x, \mathbf{p}))$ vanishes at the boundary. The purpose of this section is to illustrate how one might choose to construct these two functions depending on the form of the differential equation in question. Consider the following basic initial-value problem:

$$\frac{du(x)}{dx} = f(x, u), \quad x \in [0, 1], \quad u(0) = A \tag{30}$$

Here, we might choose the trial solution to be

$$u_t(x, \mathbf{p}) = A + x\mathbf{N}(x, \mathbf{p}) \tag{31}$$

Clearly, $A(x) = A$ satisfies the constraint imposed by the initial condition and $x\mathbf{N}(x, \mathbf{p})$ contributes nothing when $x = 0$. Moreover, this form of the trial solution and its derivative with respect to the only input x allows for easy differentiation with respect to the network parameters using equations (27), (28), (29). Choosing a uniform discretization of the interval $[0, 1]$ by the evenly spaced grid points x_i, we can write an expression for the loss function obtained by subtracting the right hand side from (30) and substituting the result for K in equation (24)

$$loss(\mathbf{p}) = \sum_i \left(\frac{du_t(x_i, \mathbf{p})}{dx} - f(x_i, u_t(x_i, \mathbf{p})) \right)^2$$

Similarly, we can consider the following second-order boundary-value problem:

$$\frac{d^2u(x)}{dx^2} = f\left(x, u(x), \frac{du(x)}{dx}\right), \quad x \in [0,1], \quad u(0) = A,\ u(1) = B \tag{32}$$

Here, we may take the trial solution to be

$$u_t(x, \mathbf{p}) = A(1-x) + Bx + x(1-x)\mathbf{N}(x, \mathbf{p}) \tag{33}$$

where once again, taking $A(x) = A(1-x) + Bx$ satisfies the boundary conditions and $x(1-x)\mathbf{N}(x, \mathbf{p})$ contributes nothing at the boundary. Subtracting $f\left(x, u, \frac{du}{dx}\right)$ from (32) and taking the left-hand side to be K as before, we obtain the loss function:

$$loss(\mathbf{p}) = \sum_i \left(\frac{d^2u_t(x_i, \mathbf{p})}{dx^2} - f\left(x_i, u_t(x_i, \mathbf{p}), \frac{du_t(x_i, \mathbf{p})}{dx}\right)\right)^2$$

The final case that we consider is that of two-dimensional PDEs with Dirichlet boundary conditions, such as the following Poisson equation:

$$\frac{\partial^2u(x,y)}{dx^2} + \frac{\partial^2u(x,y)}{dy^2} = f(x,y), \quad x \in [0,1],\ y \in [0,1]$$

subject to the boundary conditions $u(0,y) = f_0(y)$, $u(1,y) = f_1(y)$, $u(x,0) = g_0(x)$, $u(x,1) = g_1(x)$. Here, we use the trial solution

$$u_t(x, y, \mathbf{p}) = A(x,y) + x(1-x)y(1-y)\mathbf{N}(x, y, \mathbf{p}) \tag{34}$$

where the first term in the sum, $A(x,y)$, is now a function of two variables and is chosen to satisfy the boundary conditions and $x(1-x)y(1-y)\mathbf{N}(x,y,\mathbf{p})$ vanishes at the boundary. The construction that we will be using is

$$A(x,y) = (1-x)f_0(y) + xf_1(y) + (1-y)[g_0(x) - (1-x)g_0(0) - xg_0(1)] + y[g_1(x) - (1-x)g_1(0) - xg_1(1)] \tag{35}$$

Following the previous examples we choose grid points x_i that partition $[0,1] \times [0,1]$ into equal squares and substitute K in (24) by the residual error found by substituting the trial function into the differential equation. This $loss$ is thus given by

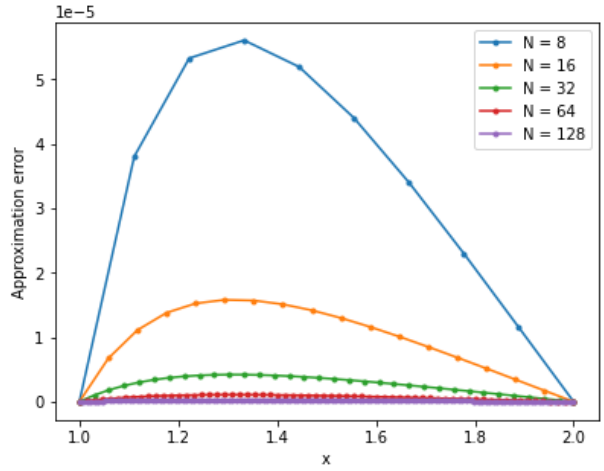
$$loss(\mathbf{p}) = \sum_i \left(\frac{\partial^2u_t(x_i, y_i, \mathbf{p})}{\partial x^2} + \frac{\partial^2u(x_i, y_i, \mathbf{p})}{\partial y^2} - f(x_i, y_i)\right)^2$$

3 Numerical Results

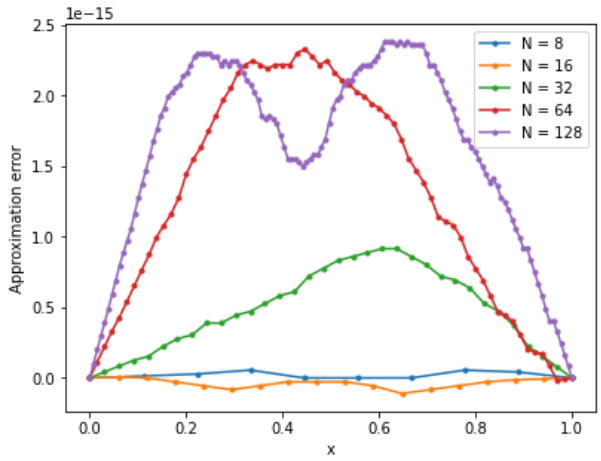
The following four sections present numerical results illustrating the theory above. We give examples of linear and nonlinear finite-difference in sections 3.1 and 3.2, where we both plot the approximation error at the grid points and the l_∞ error as we decrease the subinterval spacing. Section 3.3 gives two examples of Poisson equation in a square domain that are solved using 2D FDM. Section 3.4 highlights the difference in using piecewise-linear or splines bases in the Ritz-Galerkin method using plots of the approximation error (of the interpolating function) and error convergence plots. Finally, 3.5 and 3.6 give several examples highlighting the method of Lagaris et al. [5] using deep learning applied to 1D and 2D cases and section 3.7 provides a comparison of each method by a plot of the l_∞ error against the degrees of freedom.

3.1 Linear Finite-Differences

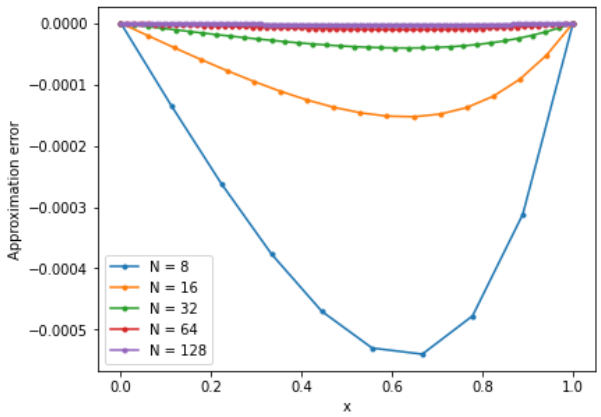
To illustrate the method of section 2.1, we give four examples in which we plot the accuracy at the grid points of five approximate solutions, denoted in the plots as $\tilde{u}$, and plot the convergence of the l_∞ error corresponding to $N = 8, 16, 32, 64, 128$:



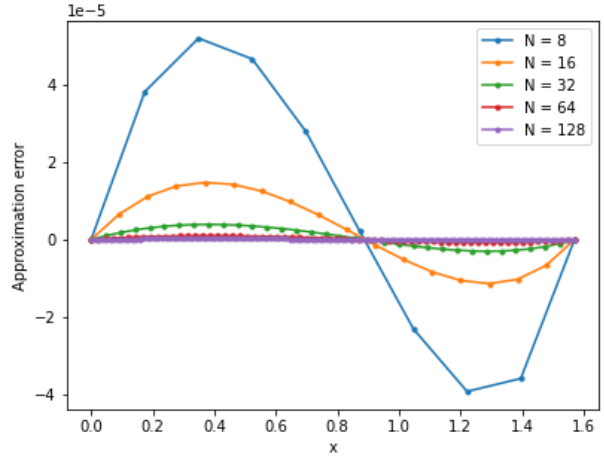
(a) $u'' = -\frac{2}{x}u' + \frac{2}{x^2}u + \frac{\sin(\ln x)}{x^2}$



(b) $u'' = -2$



(c) $u'' = 4(u - x)$



(d) $u'' = u' + 2u + \cos(x)$

Figure 4: Approximation error $u - \tilde{u}$ at the grid points of the approximate solution to the ODE for several values of N

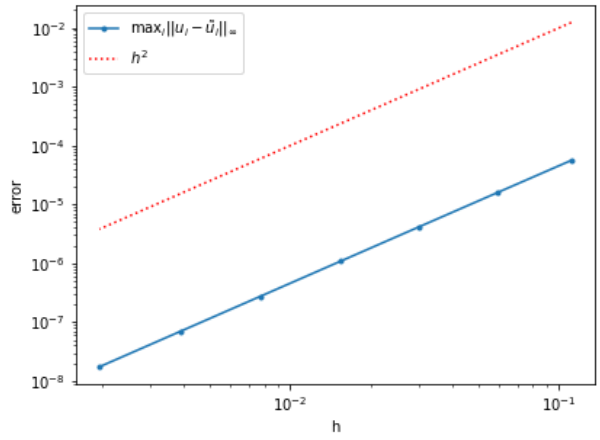
3.1.1 Example 1

$$u''(x) = -\frac{2}{x}u'(x) + \frac{2}{x^2}u(x) + \frac{\sin(\ln x)}{x^2}, \quad x \in [1, 2], \text{ with } u(1) = 1 \text{ and } u(2) = 2 \tag{36}$$

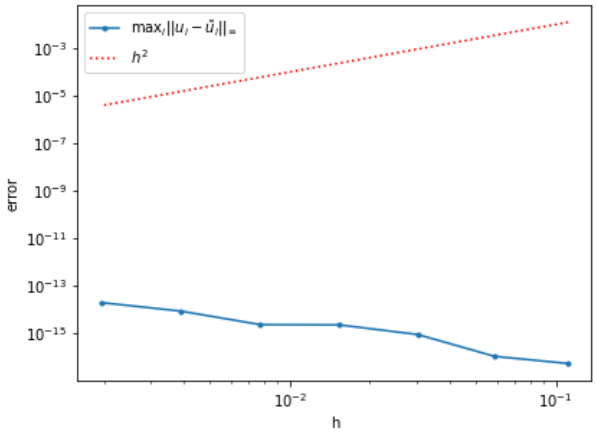
(From Burden et al. [1, p.703]) Using the algorithm of section 2.1 with variable grid spacing, we solve the linear system in (8) to get a sequence of approximations for the exact solution u at the grid points. The exact solution to this ODE is

$$u(x) = \left(\frac{1}{70x^2}\right)\left(69x^3 + 12x^3\sin(\ln(2)) + 4x^3\cos(\ln(2)) - 21x^2\sin(\ln(x)) - \right.$$
$$\left. -7x^2\cos(\ln(x)) + 8 - 12\sin(\ln(2)) - 4\cos(\ln(2))\right)$$

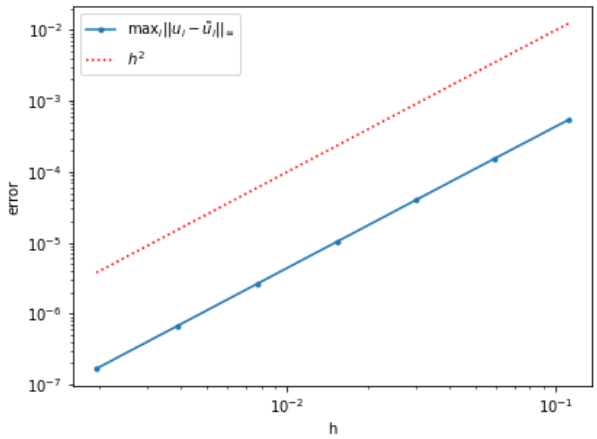
Plotting the accuracy at the grid points of the five approximate solutions gives the result in Figure 4a. Figure 5a illustrates the rate of convergence of the l_∞ error at the grid points on a log-log plot as the subinterval spacing tends to 0. The latter result demonstrates a rate of convergence of $O(h^2)$, which agrees with the truncation error.



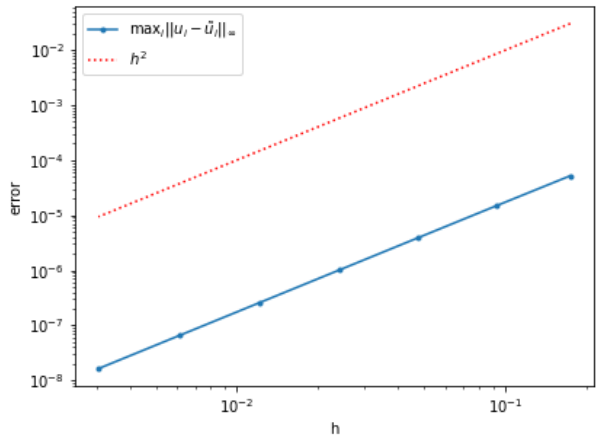
(a) $u'' = -\frac{2}{x}u' + \frac{2}{x^2}u + \frac{\sin(\ln x)}{x^2}$



(b) $u'' = -2$



(c) $u'' = 4(u - x)$



(d) $u'' = u' + 2u + \cos(x)$

Figure 5: Rate of convergence of the error at the grid points as h tends to zero

3.1.2 Example 2

The 1D Poisson equation with $f(x) = -2$:

$$u''(x) = -2, \quad x \in [0, 1], \text{ with } u(0) = u(1) = 0$$

This equation has solution $u(x) = x(1 - x)$, which has no truncation error in the centred difference formulas 1 and 7. Thus, the approximations at the grid points are equal to the value of the solution there. This explains the results of Figures 4b and 5b, which show very small error values that are at the level of machine precision.

3.1.3 Example 3

Consider now the following boundary value problem,

$$u''(x) = 4(u - x), \quad x \in [0, 1], \text{ with } u(0) = 0 \text{ and } u(1) = 2$$

(Exercise 1 in Burden et al. [1, p.704]) which has exact solution $u(x) = e^2(e^4 - 1)^{-1}(e^{2x} - e^{-2x}) + x$. Again, using the technique of section 2.1 with variable subinterval spacing, we plot the solution accuracy and the rate of convergence in Figures 4c and 5c. Notably, the latter is again observed to be $O(h^2)$.

3.1.4 Example 4

Consider this final example for the linear finite-difference case with a different choice of endpoints:

$$u''(x) = u'(x) + 2u(x) + \cos(x), \quad x \in \left[0, \frac{\pi}{2}\right], \text{ with } u(0) = -0.3 \text{ and } u\left(\frac{\pi}{2}\right) = -0.1$$

(Exercise 2 in Burden et al. [1, p.704]) This BVP has solution $u(x) = -\frac{1}{10}(\sin(x) + 3\cos(x))$. Using the method of 2.1, a numerical approximation at the grid points was calculated for various values of h. The accuracy of the first five solutions and the convergence of the error is visible in Figures 4d and 5d, the latter demonstrating a rate of convergence of $O(h^2)$.

3.2 Nonlinear Finite-Differences

Now, we present four examples of the finite difference method applied to nonlinear ODEs as presented in section 2.1.2 using Newton's method. Thus, in each example, we produce a sequence of iterates that approximates the solution at the grid points with increasing accuracy for fixed h as in equation (12), using a relative error stopping criterion of 10^{-8}. As in the previous section, we consider 5 representative approximations corresponding to the values $N = 8, 16, 32, 64, 128$ and plot the solution accuracy at the grid points for each approximation, denoted $\tilde{u}$, as well as the l_∞ norm on a log-log plot.

3.2.1 Example 5

Consider, first, the following boundary value problem that is nonlinear in u.

$$u''(x) = \frac{1}{8}(32 + 2x^3 - u(x)u'(x)), \quad \text{for } x \in [1, 3], \text{ with } u(1) = 17 \text{ and } u(3) = \frac{43}{3}$$

(Example 1 in Burden et al. [1, p.710]) The exact solution is $u(x) = x^2 + 16/x$. Repeating Newton's method for increasingly small values of h yields the solution accuracy at the grid points shown in Figure 6a. The rate of convergence is shown to be $O(h^2)$ in Figure 7a.

3.2.2 Example 6

As a second example, consider the following nonlinear boundary value problem,

$$u''(x) = -u'(x)^2 - u(x) + \ln x, \quad x \in [1, 2], \text{ with } u(1) = 0 \text{ and } u(2) = \ln 2$$

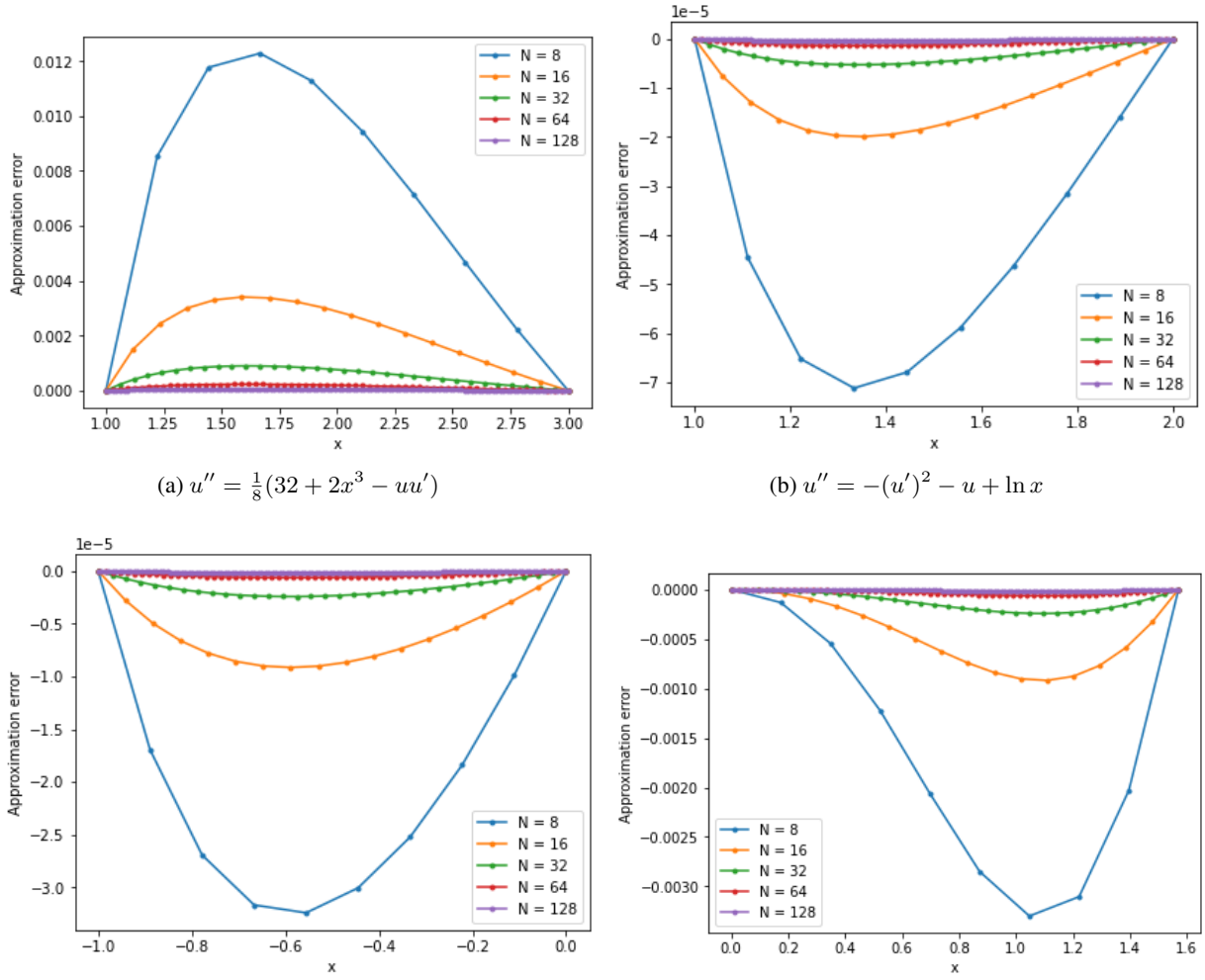
(Exercise 1 in Burden et al. [1, p.711]) with exact solution $u(x) = \ln(x)$. Again, using the methods of section 2.1.2, we produce Figures 6b and 7b of the solution accuracy at the grid points for 5 values of the subinterval spacing h and the rate of convergence of the l_∞ norm as h tends to 0. The latter is shown to be $O(h^2)$ as before.

3.2.3 Example 7

Consider the following nonlinear boundary value problem with a negative endpoint:

$$u''(x) = 2u(x)^3, \quad x \in [-1, 0], \text{ with } u(-1) = \frac{1}{2} \text{ and } u(0) = \frac{1}{3}$$

(Exercise 2 in Burden et al. [1, p.711]) This problem has solution $u(x) = 1/(x + 3)$. The solution accuracy at the grid points is given for the first 5 values of h in Figure 6c and the rate of convergence is shown to be $O(h^2)$ in Figure 7c.



(a) $u'' = \frac{1}{8}(32 + 2x^3 - uu')$

(b) $u'' = -(u')^2 - u + \ln x$

(c) $u'' = 2u^3$

(d) $u'' = u' \cos x - u \ln u$

Figure 6: Approximation error $u - \tilde{u}$ at the grid points of the approximate solution to the ODE

3.2.4 Example 8

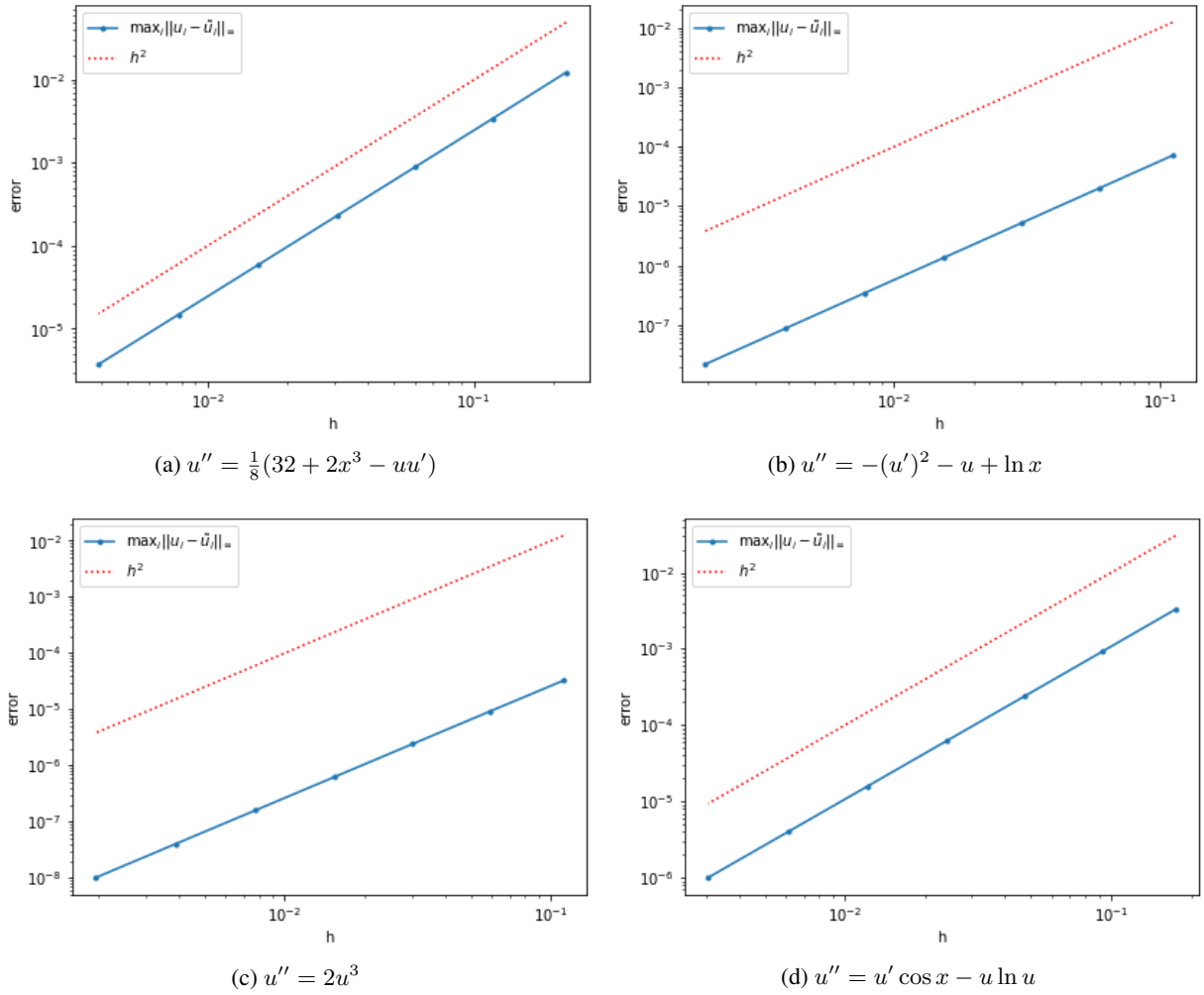
The final example featuring cosines and natural logarithms,

$$u''(x) = u'(x)\cos x - u(x)\ln u(x), \quad x \in \left[0, \frac{\pi}{2}\right], \text{ with } u(0) = 1 \text{ and } u\left(\frac{\pi}{2}\right) = e,$$

(Exercise 3.b in Burden et al. [1, p.711]) has solution $u(x) = e^{\sin x}$ with plots of approximate solution accuracy and $O(h^2)$ rate of convergence shown in Figures 6d and 7d.

3.3 2D Poisson Equation using finite-differences

To illustrate the method of section 2.1.3, we solve two similar boundary-value problems taken from Lagaris et al. [5].



(a) $u'' = \frac{1}{8}(32 + 2x^3 - uu')$

(b) $u'' = -(u')^2 - u + \ln x$

(c) $u'' = 2u^3$

(d) $u'' = u' \cos x - u \ln u$

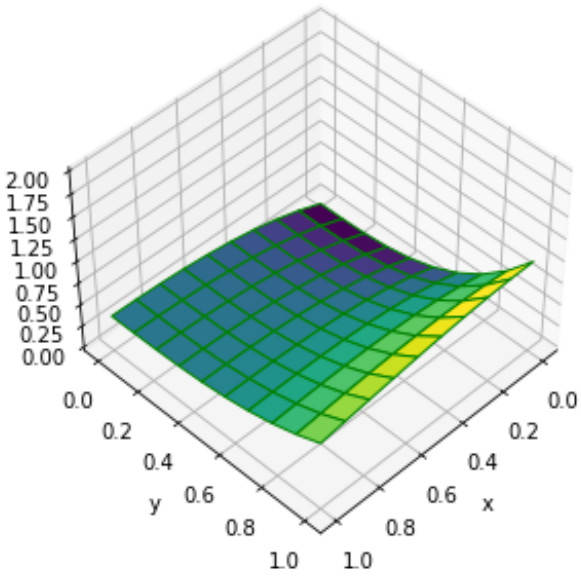
Figure 7: Rate of convergence of the error at the grid points as h tends to zero

3.3.1 Example 9

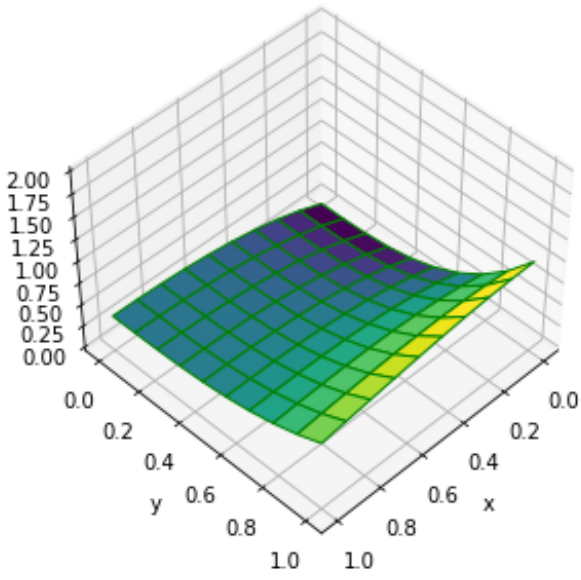
Consider the following boundary-value problem with Dirichlet boundary conditions:

$$\frac{\partial^2 u}{\partial x^2}(x, y) + \frac{\partial^2 u}{\partial y^2}(x, y) = e^{-x}(x - 2 + y^3 + 6y), \quad x, y \in [0, 1] \tag{37}$$

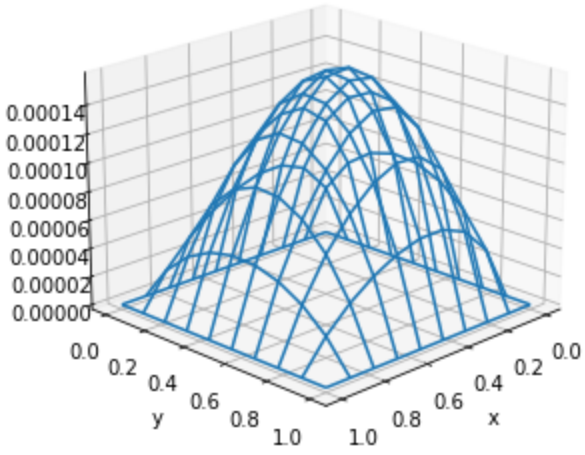
(Problem 5 in Lagaris et al. [5]) This equation has exact solution $u(x, y) = e^{-x}(x + y^3)$ and the boundary conditions are specified by substituting x and y with 0 and 1, namely $f_0(y) = u(0, y) = y^3$, $f_1(y) = u(1, y) = (1 + y^3)e^{-1}$, $g_0(x) = u(x, 0) = xe^{-x}$, and $g_1(x) = u(x, 1) = e^{-x}(x + 1)$. Figure 8 provides a comparison between the exact solution and the approximate solution, consisting of a matrix of 100 values at the grid points. Note that the error, shown in 8c, is zero at the boundary since the approximate solution satisfies the boundary conditions defined by the exact solution. The performance illustrated in this figure indicates that the absolute error is lower close to the boundary and reaches its maximum of 1.591×10^{-4} in the centre.



(a) Exact solution



(b) Approximate solution



(c) Approximation error at the grid points

Figure 8: Finite-difference approximation for Example 9

3.3.2 Example 10

The next boundary-value problem is similar to the last, but with a more complicated expression for $f(x, y)$ containing sines and cosines:

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = e^{-\frac{3x+y}{5}} \left[\left(-\frac{108}{5}x + \frac{88}{5} \right) \cos(9x^2 + y) + \left(324x^2 - \frac{15}{25} \right) \sin(9x^2 + y) \right] \tag{38}$$

(Problem 6 in Lagaris et al. [5]) where $x, y \in [0, 1]$ and the exact solution is given by $u(x, y) = e^{-\frac{3x+y}{5}} \sin(9x^2 + y)$. As before, the boundary conditions are defined by taking the exact solution at the boundary, that is $f_0(y) = u(0, y) = e^{\frac{y}{5}} \sin y$, $f_1(y) = u(1, y) = e^{-\frac{3+y}{5}} \sin(9 + y)$, $g_0(x) = u(x, 0) = e^{-\frac{3x}{5}} \sin(9x^2)$, and $g_1(x) = u(x, 1) = e^{-\frac{3x+1}{5}} \sin(9x^2 + 1)$. The results of the finite-difference approximation are illustrated in Figure 9. This time, Figure 9c indicates that the absolute error is greater close to the boundary $x = 1$, where it reaches its maximum value of 0.1185.

3.4 Ritz-Galerkin

To illustrate the method of section 2.2, we present two examples, each using both the piecewise-linear and bell-shaped splines bases.

3.4.1 Example 11

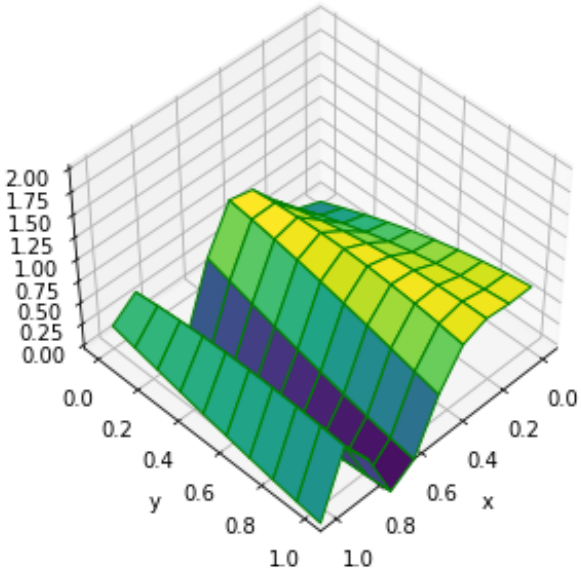
$$-u''(x) + \pi^2 u(x) = 2\pi^2 \sin(\pi x), \quad x \in [0, 1], \text{ with } u(0) = u(1) = 0$$

(Illustration in Burden et al. [1, p.719]) This second order linear BVP has exact solution $u(x) = \sin \pi x$. Figures 10a and 10c show the accuracy of the solution approximations in the piecewise-linear and splines bases, respectively, at various choices of the grid points. It is immediately clear from these plots that the splines basis is more accurate overall, with significantly smaller error values for each choice of grid point spacing. In Figures 11a and 11c, we see that the rates of convergence of the approximate solution are $O(h^2)$ and $O(h^4)$. Thus, the method employing a splines basis enjoys a significantly faster rate of convergence.

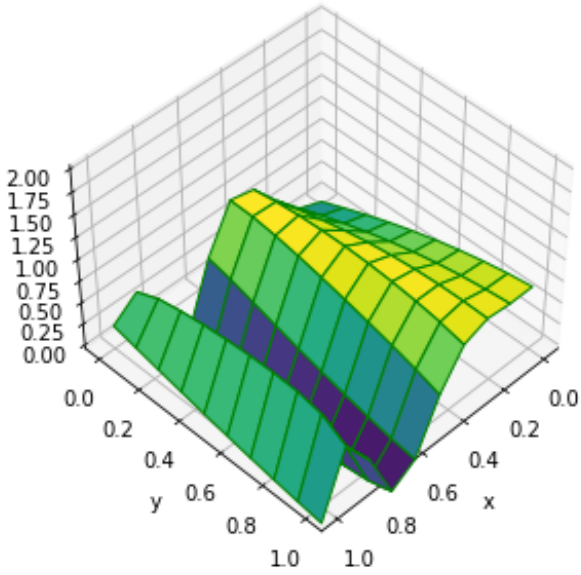
3.4.2 Example 12

$$u''(x) + \frac{\pi^2}{4} u(x) = \frac{\pi^2}{16} \cos \frac{\pi}{4} x, \quad x \in [0, 1], \text{ with } u(0) = u(1) = 0$$

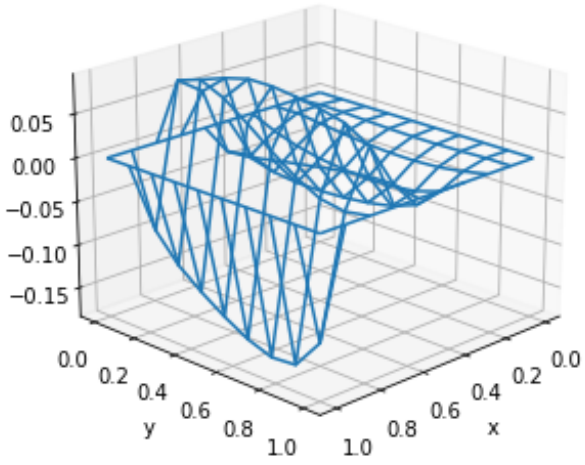
(Exercise 1 in Burden et al. [1, p.726]) The exact solution to this BVP is $u(x) = -\frac{1}{3} \cos \frac{\pi}{2} x - \frac{\sqrt{2}}{6} \sin \frac{\pi}{2} x + \frac{1}{3} \cos \frac{\pi}{4} x$. The accuracy of the approximate solutions for various choices of the grid points is plotted in Figures 10b and 10d, corresponding to the piecewise-linear and splines bases, respectively. The rates of convergence for the two choices of basis are plotted in Figures 11b and 11d for several choices of h. Again, the results show significantly smaller error at the grid points and in the interpolated regions when splines are used. As in the previous example, the results corroborate the theory section, which predicted the splines basis converging faster, with an actual rate of $O(h^4)$.



(a) Exact solution

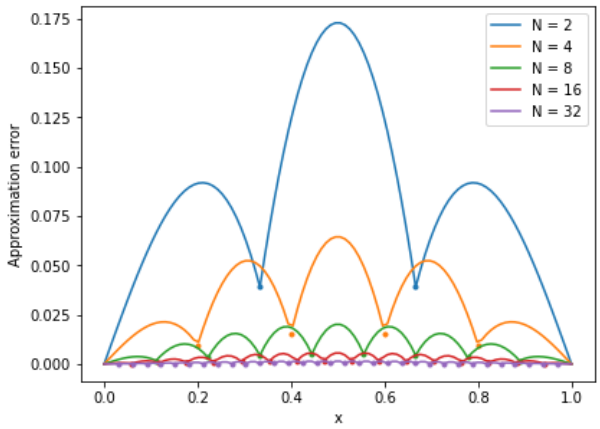


(b) Approximate solution

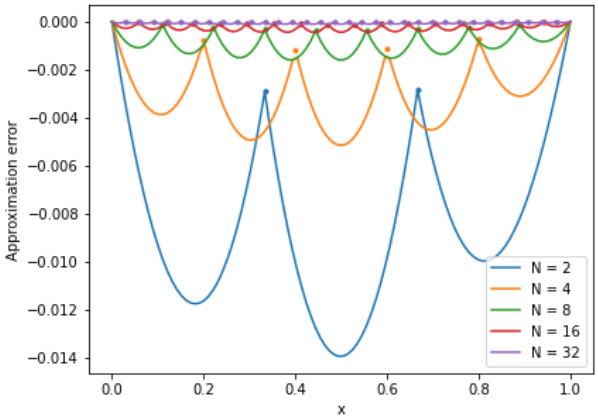


(c) Approximation error at the grid points

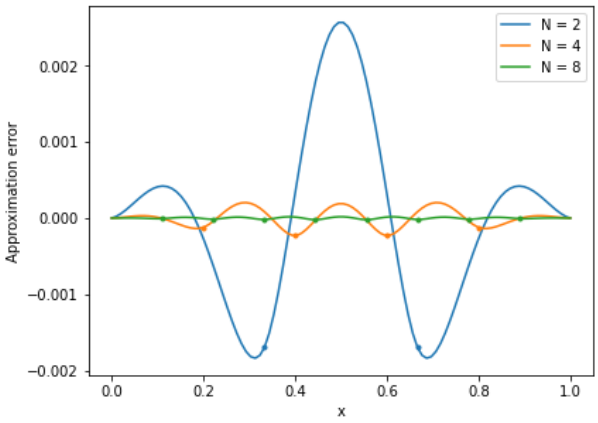
Figure 9: Finite-difference approximation for Example 10



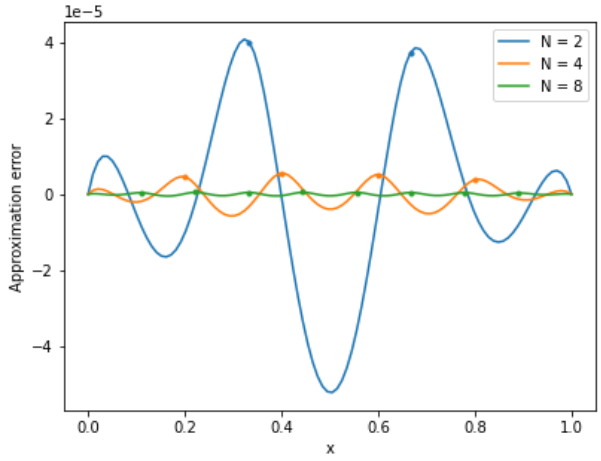
(a) Example 11 with piecewise-linear basis



(b) Example 12 with piecewise-linear basis

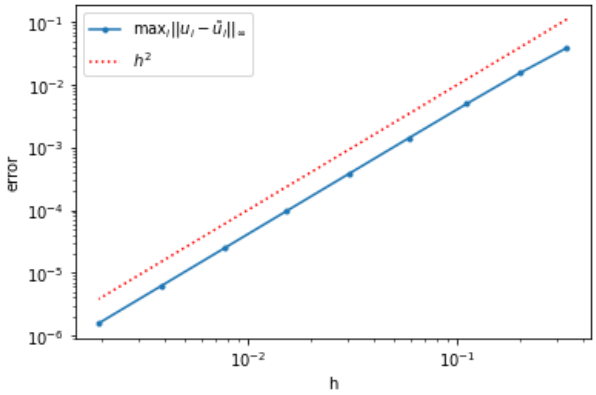


(c) Example 11 with bell-shaped splines basis

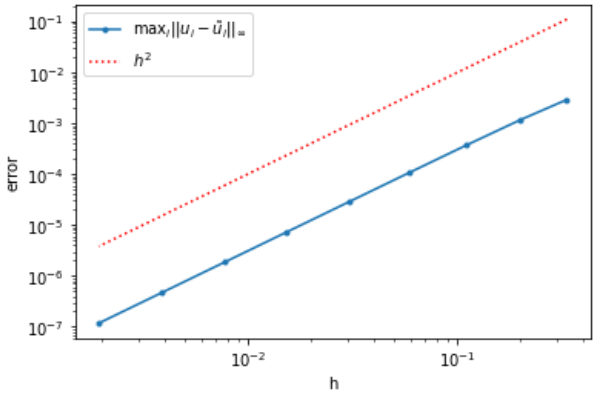


(d) Example 12 with bell-shaped splines basis

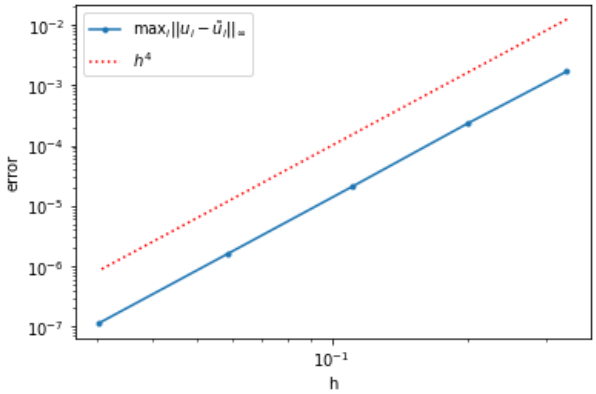
Figure 10: Approximation error $u - \tilde{u}$ of the approximate solution $\tilde{u}$ to the ODE for various choices of N.



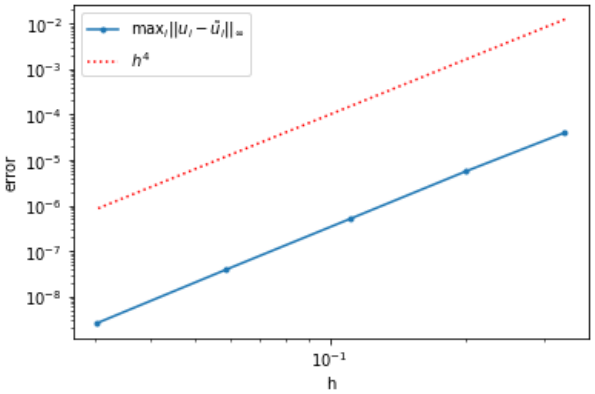
(a) Example 11 with piecewise-linear basis



(b) Example 12 with piecewise-linear basis



(c) Example 11 with bell-shaped splines basis



(d) Example 12 with bell-shaped splines basis

Figure 11: Rate of convergence of the error at the grid points as h tends to zero

3.5 1D PINN

In this section, we provide three examples taken from Lagaris et al. [5] to illustrate the method of section 2.3. These examples are ordinary differential equations with Dirichlet boundary conditions that are all trained using the model detailed in section 2.3 using 10 hidden sigmoid neurons and a linear output neuron. Each trial is performed with a learning period of 10,000 epochs using the Adam optimizer that is built into Keras [2][4] with an initial learning rate of 0.1 that decays linearly to 10^{-4}.

3.5.1 Example 13

$$u'(x) + \left(x + \frac{1+3x^2}{1+x+x^3}\right) u(x) = x^3 + 2x + x^2 \frac{1+3x^2}{1+x+x^3}, \quad x \in [0,1], \text{ with } u(0) = 1$$

(Problem 1 in Lagaris et al. [5]) This first example is a first-order ODE with a single Dirichlet boundary condition or initial condition. The exact solution is given by $u(x) = \frac{e^{-x^2/2}}{1+x+x^3} + x^2$ and is referred to as "target" in Figure 12a. Using equation (31), we write the trial solution as $u_t(x, \mathbf{p}) = 1 + xN(x, \mathbf{p})$, which satisfies the initial condition. Using, the model in section 2.3, we produce the results in Figure 12. Figure 12a also illustrates the ability of this method to extrapolate to points outside the interval $[0, 1]$, which is done with some degree of accuracy. Figure 12c showcases the accuracy of the trained model in the interval where grid points are taken.

3.5.2 Example 14

$$u'(x) + \frac{1}{5}u(x) = e^{-x/5}\cos(x), \quad x \in [0,2], \text{ with } u(0) = 0$$

(Problem 2 in Lagaris et al. [5]) Now, we consider once again a first-order ODE, but with a cosine function on the right-hand side and a larger interval for x. The exact solution is given by $u(x) = e^{-x/5}\sin(x)$ and is plotted in Figure 13a against the trial solution given by $u_t(x, \mathbf{p}) = xN(x, \mathbf{p})$ for points in and also outside the interval $[0, 2]$. The accuracy of the trained model in this interval is illustrated in Figure 13c.

3.5.3 Example 15

$$u''(x) + \frac{1}{5}u'(x) + u(x) = -\frac{1}{5}e^{-\frac{x}{5}}\cos x, \quad x \in [0,1], \text{ with } u(0) = 0 \text{ and } u(1) = \sin(1)e^{-\frac{1}{5}}$$

(Problem 3 in Lagaris et al. [5]) This next example is a second-order ODE. The exact solution is given by $u(x) = e^{-\frac{x}{5}}\sin(x)$ and the trial solution is $u_t(x, \mathbf{p}) = x\sin(1)e^{-\frac{1}{5}} + x(1-x)N(x, \mathbf{p})$ (from equation (33)). Using, the same model, we produce the results in Figure 14.

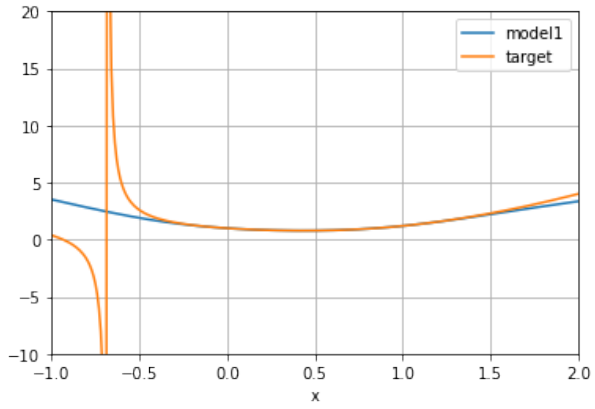
3.6 2D PINN

The following two examples are actually the same as examples 9 and 10, but now the approximate solutions are functions of the respective outputs of trained neural networks, using the architecture of section 2.3. In these examples, a single hidden layer composed of 10 sigmoid neurons is used, along with a linear output and 2 input neurons. Again, each trial is performed with a learning period of 10,000 epochs using the built-in Adam optimizer with an initial learning rate of 0.1 that decays linearly to 10^{-4}.

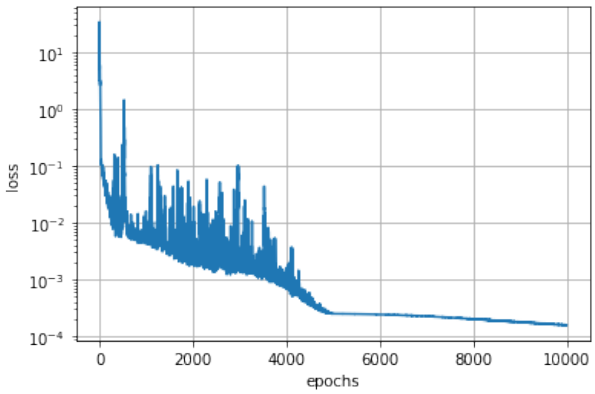
3.6.1 Example 16

From equation (34), we construct the trial solution for the PDE (37), composed of a first term $A(x, y)$ that satisfies the boundary conditions and a second term, consisting of the model output, that vanishes at the boundary:

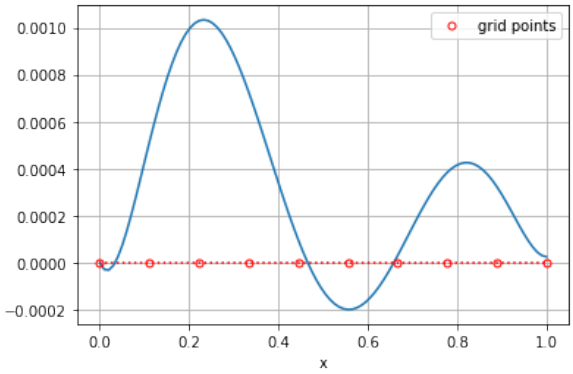
$$u_t(x, y, \mathbf{p}) = A(x, y) + x(1-x)y(1-y)N(x, y, \mathbf{p})$$



(a) Plot of the test solution against the target solution



(b) Decay of the residual error (loss) in each epoch



(c) Approximation error of the model

Figure 12: Result of training the PINN model1 for Example 13

Using equation (35), we substitute the boundary conditions and obtain an explicit formula for $A(x, y)$:

$$A(x,y) = (1-x)y^3 + e^{-1}x(1+y^3) + (1-y)[xe^{-x} - xe^{-1}] + y[e^{-x}(x+1) - (1-x) - 2xe^{-1}]$$

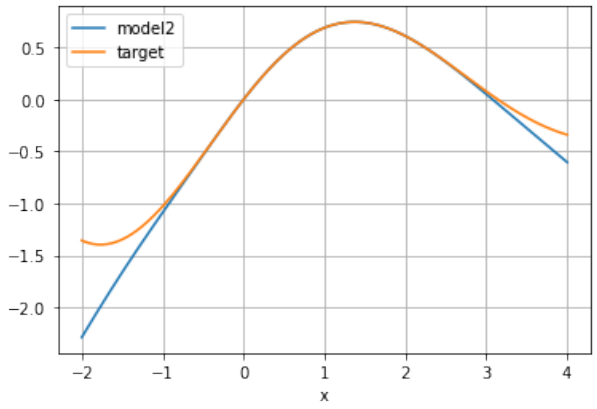
Figure 15 illustrates the results of the model. The accuracy at the grid points is shown in Figure 16b and Figure 16c depicts the learning history. As in section 3.3, the error is observed to be zero at the boundary since the trial solution is constructed to satisfy the boundary conditions. The maximum absolute error is found to be 8.069×10^{-3}.

3.6.2 Example 17

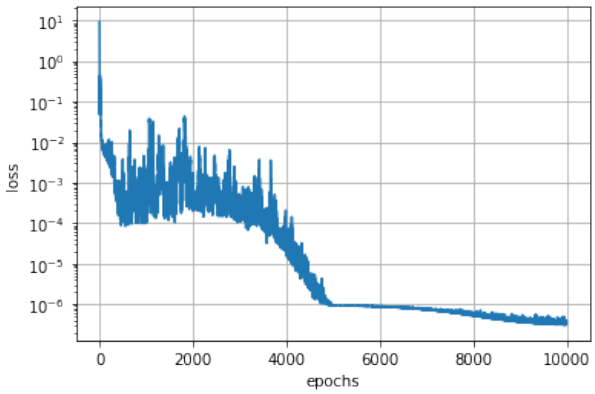
Again, we use equation (34) to write the trial solution $u_t(x, y, \mathbf{p}) = A(x, y) + x(1-x)y(1-y)N(x, y, \mathbf{p})$ for the PDE (38) and use equation (35) with the boundary conditions specified as in Example 10 to write an explicit form for the first term $A(x, y)$:

$$\begin{aligned} A(x,y) = (1-x)e^{-\frac{y}{5}}\sin y + xe^{-\frac{3+y}{5}}\sin(9+y) + (1-y)\left[e^{-\frac{3x}{5}}\sin(9x^2) - xe^{-\frac{3}{5}}\sin 9\right] \\ + y\left[e^{-\frac{3x+1}{5}}\sin(9x^2+1) - (1-x)e^{-\frac{1}{5}}\sin 1 - xe^{-\frac{4}{5}}\sin 10\right] \end{aligned}$$

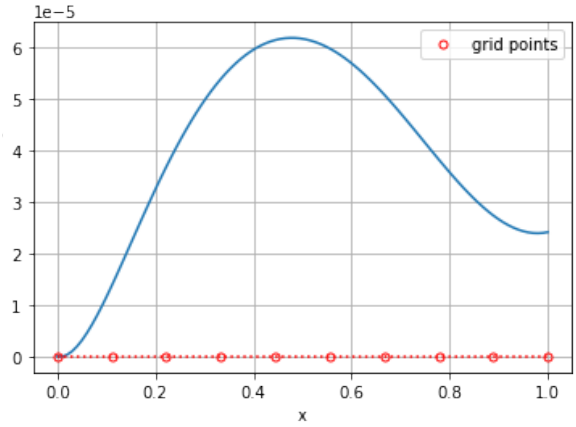
The results of the learning are depicted in Figure 16, where the maximum absolute error is found to be 2.725×10^{-2}, which is markedly lower than the corresponding measurement taken using finite differences in Example 10.



(a) Plot of the test solution against the target solution



(b) Decay of the residual error (loss) in each epoch



(c) Approximation error of the model

Figure 13: Result of training the PINN model2 for Example 14

3.7 Comparisons

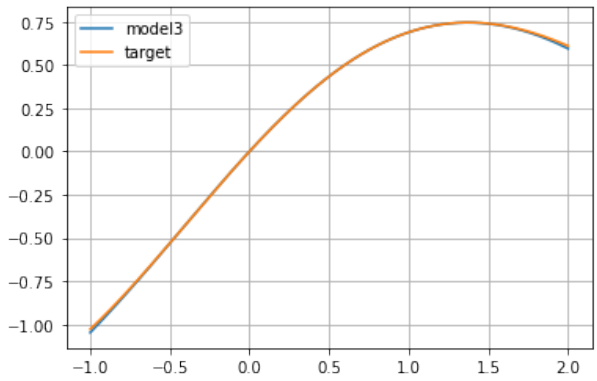
We aim to determine whether the results suggest a case for using deep learning to solve PDE over the various classical methods. To accomplish this, we next present two examples, one for the 1D case and one for the 2D case, and compare the results obtained by varying the number of degrees of freedom for each of the methods described.

3.7.1 1D comparison

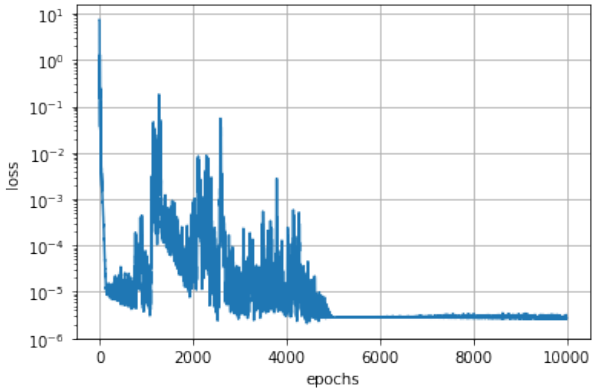
To compare the methods used to approximate single dimensional ordinary differential equations, we use the following boundary-value problem (Exercise 3.d from Burden et al. [1] p. 726) that can be written in either of the forms discussed in sections 2.1.1 and 2.2:

$$-(x+1)u''(x) - u'(x) + (x+2)u(x) = (2-(x+1)^2)e\ln 2 - 2e^x, \quad x \in [0,1], \text{ with } u(0) = u(1) = 0 \tag{39}$$

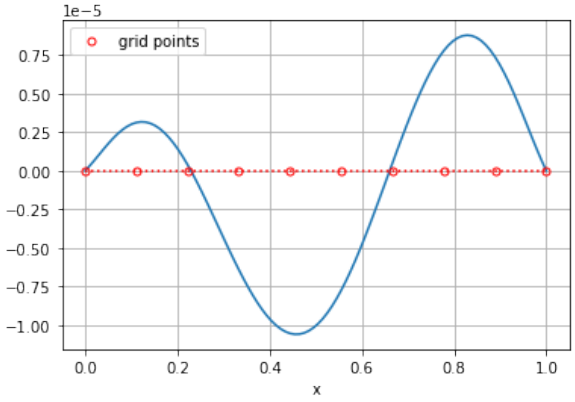
This second-order linear ODE has exact solution $u(x) = e^x \ln(x+1) - (e\ln 2)x$ and can be solved by using finite-differences, Ritz-Galerkin method with either the piecewise-linear or splines bases, and deep learning. The results of the training with 10 hidden neurons are illustrated in Figure 17. In finite-differences, the degrees of freedom (dof) is taken as the dimensionality of the linear system, N. In Ritz-Galerkin, the dof, or the number of coefficients to solve for, depends on the choice of basis: N for the piecewise-linear basis, and $N+2$ for the splines. Finally, for deep learning,



(a) Plot of the test solution against the target solution



(b) Decay of the residual error (loss) in each epoch



(c) Approximation error of the model

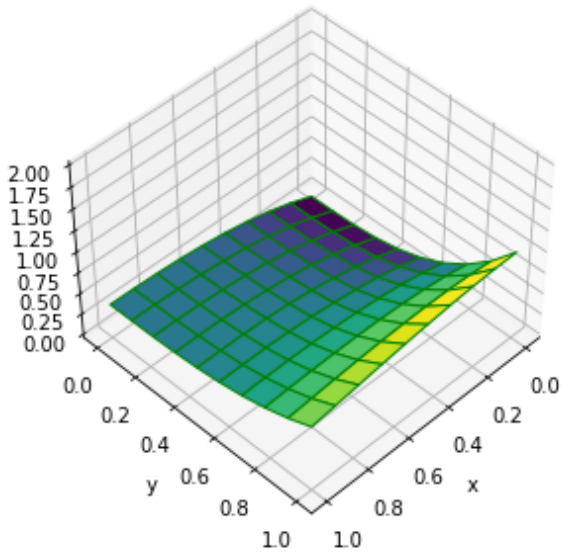
Figure 14: Result of training the PINN model3 for Example 15

we chose to fix the depth of the network to one hidden layer and instead vary the width of the single hidden layer by adding neurons. For example, keeping only one hidden neuron gives a network with 3 degrees of freedom since there is only one weighted input to the linear output neuron, one weighted input to the hidden neuron, and the latter's implicit bias. If we keep two hidden neurons, we get 6 degrees of freedom since there are now two inputs to the linear output neuron, an additional weighted input to the second hidden neuron, plus the latter's implicit bias.

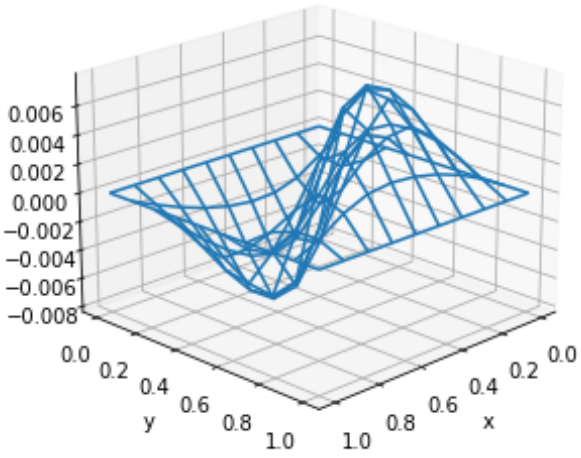
Figure 18 illustrates the difference in using each of the methods by plotting the l_∞ error versus the number of degrees of freedom, which is taken at 5 values: 6, 9, 12, 24, 48, 96. This plot presents two striking qualities. Firstly, the error from the deep learning model is drastically reduced, even becoming the lowest l_∞ error among the methods considered, as we increase the dof to 9. Secondly, the model performs more or less the same, perhaps slightly worse, as the dof is increased further. This second feature may have multiple causes, among them being the increased cost of having to train extra free parameters in the neural network. Another possible explanation, that is not related to the network *architecture*, could be that the training procedure is being hamstrung by a bottleneck in the learning *algorithm*, perhaps in the choice of optimizer.

3.7.2 2D comparison

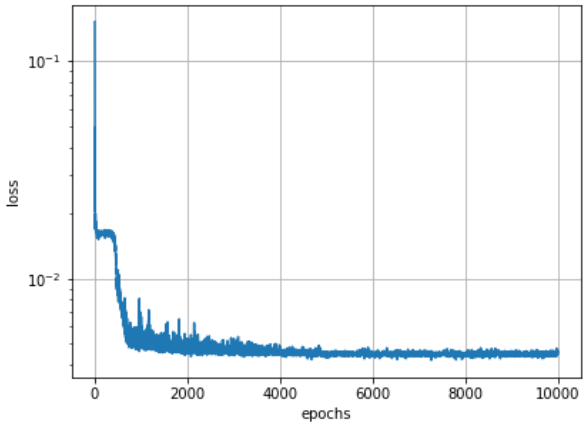
To compare the methods used to approximate the 2D PDE, we reuse Example 10 from section 2.1.3. In the case of finite-differences, the dof is still given by the dimensionality of the linear system, but now this feature is a product of the number of inner grid points, that is, $(M-1) \times (N-1)$. In deep learning, we once again fix the depth of the network



(a) Trial solution



(b) Approximation error at the grid points

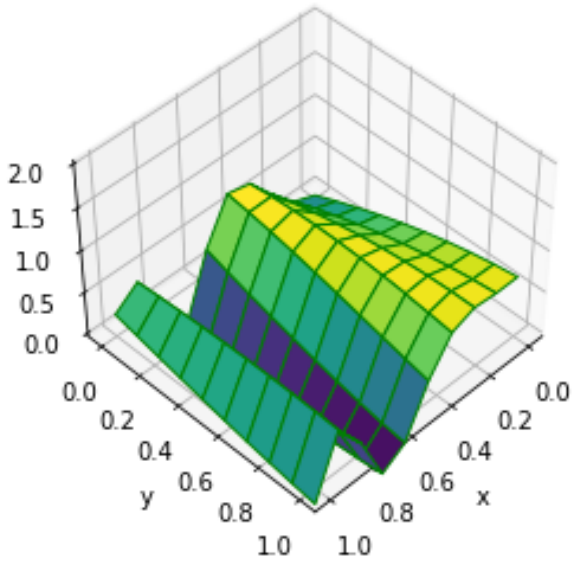


(c) Decay of the residual error (loss)

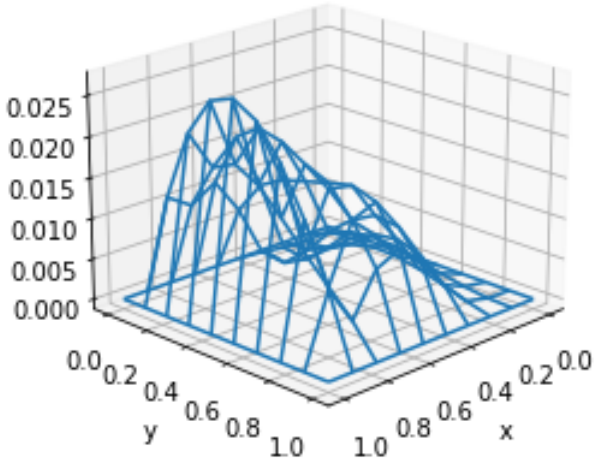
Figure 15: Result of the training for Example 16

at 1 hidden layer and vary the number of neurons in this layer. Counting the degrees of freedom is as before, except that now there is an extra input node, which doubles the weights to the hidden layer. For example, if $H = 1$, then there are now two weighted inputs to the single hidden neuron. Adding the neuron's implicit bias and the weighted input to the output neuron makes 4 dof. If $H = 2$, then there are four weighted inputs to the hidden layer, two biases, and two weighted inputs to the output neuron, making 8 dof in all.

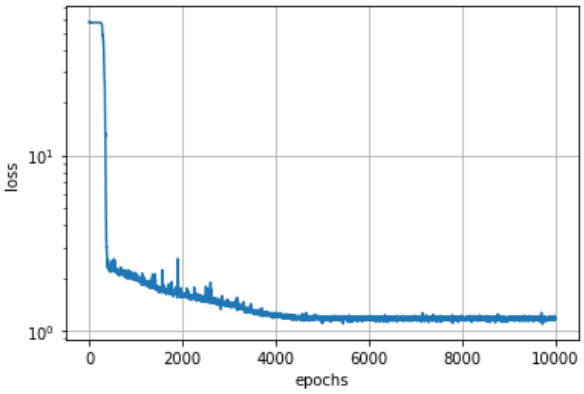
Figure 19 illustrates the effect of increasing the number of dof on the l_∞ loss incurred by using finite-differences and deep learning when applied to Example 10 for the case of 5 values of the degrees of freedom: 4, 16, 36, 64, 100. Similar qualities as in Figure 18 arise here, namely, an initially impressive error value for the deep learning model that becomes less so as more dof are added. We suspect the reasons to be as before: an increased computational cost due to the inclusion of more training parameters and, perhaps more convincingly, a poor choice of optimizer or other sub-optimal step in the training algorithm.



(a) Trial solution

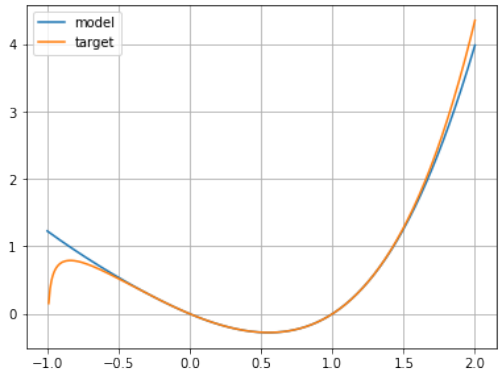


(b) Approximation error at the grid points

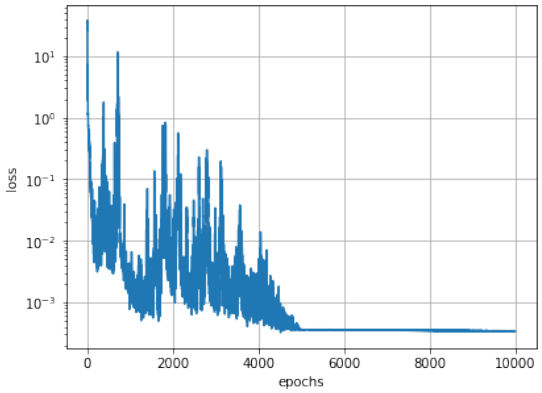


(c) Decay of the residual error (loss)

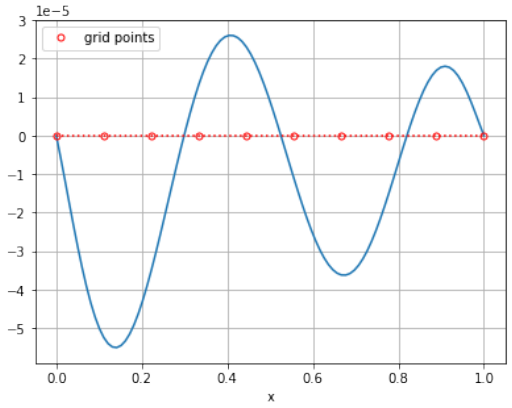
Figure 16: Result of the training for Example 17



(a) Plot of the test solution against the target solution



(b) Decay of the residual error (loss) in each epoch



(c) Approximation error at the grid points

Figure 17: Result of training the PINN model used to solve (39) using 10 hidden neurons

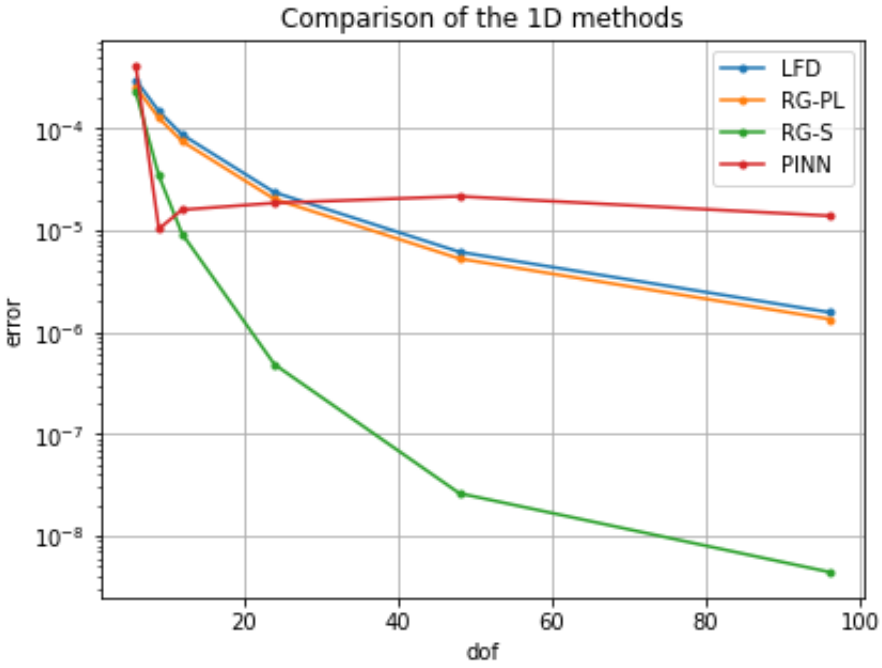


Figure 18: Effect of varying the dof on l_∞ error in equation (39) when using finite-differences, Ritz-Galerkin with piecewise-linear (PL) and splines (S) bases, and deep learning.

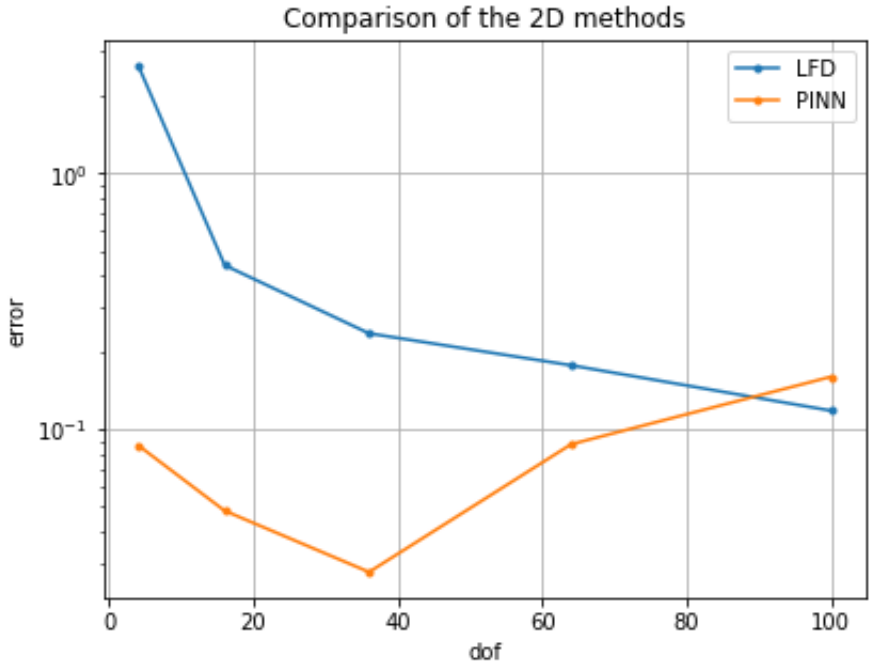


Figure 19: Effect of dof on l_∞ error in Example 10 when using finite-differences and deep learning.

4 Conclusion

In this paper, we sought to study the effects of decreasing the subinterval spacing h on the l_∞ error for solutions of ODE that were found using the methods of finite-difference and Ritz-Galerkin and compare with the theoretical values found in the literature. In fact, we were able to corroborate these values and found them to be $O(h^2)$ in the cases of finite-difference and Ritz-Galerkin with piecewise-linear basis, and $O(h^4)$ when using Ritz-Galerkin with a splines basis. In section 2.3.1, we presented an introductory account of deep learning and then proceeded in subsequent sections to depict a scenario in which physics-informed neural networks, as shown in Lagaris et al. [5], could be used to solve ordinary and partial differential equations. In sections 3.5 and 3.6, we presented several examples of deep neural networks that were able to accurately solve the differential equations therein. Finally, in section 3.7, we presented two comparisons, one in the 1D case and one for 2D, in which deep learning models were tested against the classical methods of finite-difference and Ritz-Galerkin with varying degrees of freedom.

As a final note, the results of the last section appear to indicate that although there may have been sub-optimal steps in our learning algorithm, there is clear potential for deep learning models to, at the very least, perform with much higher accuracy at lower values of the dof than the classical methods. In future work, it would be interesting to study the effects of increasing the number of hidden layers of the network, as well as the impact of using different activation functions for the hidden neurons. It would also be instructive to study the effects of using different network optimizers to potentially improve the learning. Ultimately, it would be worthwhile to document the practical implications of using deep learning instead of classical methods in industry and whether there are any clear benefits to doing so, considering also that it usually takes time to train a model until it becomes usable.

References

[1] R. L. Burden, D. J. Faires, and A. M. Burden, *Numerical Analysis*, 10th ed. Cengage Learning, 2014.

[2] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[3] C. F. Higham and D. J. Higham, "Deep learning: An introduction for applied mathematicians," *SIAM Review*, vol. 61, no. 4, pp. 860–891, 2019.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014. [Online]. Available: https://arxiv.org/abs/1412.6980

[5] I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial neural networks for solving ordinary and partial differential equations," *IEEE Transactions on Neural Networks*, 1998.

[6] S. Markidis, "The old and the new: Can physics-informed deep-learning replace traditional linear solvers?" *Frontiers in Big Data*, vol. 4, 2021. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fdata.2021.669097

[7] B. Moseley, A. Markham, and T. Nissen-Meyer, "Solving the wave equation with physics-informed deep learning," 2020. [Online]. Available: https://arxiv.org/abs/2006.11894

[8] P. Peng, J. Pan, H. Xu, and X. Feng, "Rpinns: Rectified-physics informed neural networks for solving stationary partial differential equations," *Computers & Fluids*, p. 105583, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045793022001955

[9] M. Raissi, P. Perdikaris, and G. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0021999118307125

[10] E. W. Weisstein, "Sigmoid function," https://mathworld.wolfram.com/SigmoidFunction.html, accessed: 2022-07-29.