Solution of differential equations using neural networks

Vassili Korotkine

McGill University

Structural Dynamics and Vibrations Laboratory

Supervised by Professor Mathias Legrand

A thesis submitted to McGill University in partial fulfilment of the requirements of the Undergraduate Honours Program.

Submitted Dec. 3, 2019

Abstract

A least squares approach to solving differential equations was applied to the mass spring and one-dimensional wave equations. Similarly to traditional linear least squares, an arbitrary function with some parameters was introduced. A squared error metric using the local equation was formed. The function's parameters were then found such that this squared error metric was minimized. Initial and boundary conditions were treated through the form of the arbitrary function. The proposed method worked for the mass spring system test case, but failed for the wave equation. Although the method was promising, future work remains to determine whether it can robustly be applied to complicated problems.

Une méthode des moindres carrés a été appliquée à un système masse-ressort et à l'équation d'onde en une dimension. De manière similaire aux moindres carrés linéaires classiques, une fonction arbitraire avec certains paramètres a été introduite. Une métrique d'erreur carréee utilisant l'équation locale a été formée. Les paramètres de la fonction ont ensuite été trouvés pour minimisér cette métrique d'erreur. Les conditions initiales et aux limites ont été traitées par la forme de la fonction arbitraire. La méthode proposée a fonctionné pour le système masse-resort, mais a échoué pour l'équation d'onde. Bien que la méthode ait été prometteuse, il reste encore à déterminer si elle peut être appliquée de manière robuste à des problèmes complexes.

Acknowledgements

I am grateful to Professor Mathias Legrand for his support, supervision, and having provided me with the environment to conduct this research. I would also like to thank the PhD students David Urman and Tianzheng Lu for their support and many insightful discussions. This experience would not have been the same without any of you.

Contents

Chapter 1

Introduction

Differential equations are ubiquitous in engineering applications. In particular, they are used as models for elastodynamics and heat transfer problems. A wide range of techniques exist to solve them including finite element, finite volume, and discontinuous galerkin schemes [1]. These techniques perform very well for a wide range of problems. Nevertheless, they can have issues when applied to certain problems [2, 3]. A different method to solving differential equations was investigated in this work.

Machine learning a set of techniques that create models to approximate the relationship between two sets of data (inputs and outputs). The ultimate goal is for the model to generalize to unseen data. In some cases, these models can even produce examples of new data such as in the case of generative adversarial networks [4].

A subset of machine learning techniques involves what are called neural networks. A neural network is a broad term, which can describe anything from a shallow feedforward neural network (a relatively simple composition of affine and nonlinear transformations) [5] to Long Short Term Memory Networks (LSTM's) [6] to convolutional neural networks [7]. In all cases, a model is set up with adjustable parameters that are tuned such that the model fits a desired behaviour.

In the context of machine learning, neural networks are simply nonlinear statistical models

[5]. However, they have been applied in different areas. In particular, they have been explored as a way to solve differential equations. This is not a new idea and has been around at least as far back as the 1998 paper by Lagaris et al. [8] which proposed the idea of constructing a solution ansatz with some adjustable parameters that satisfied boundary conditions. The parameters of the ansatz were found such that it fit the local equation. The idea has been explored many times since [9, 10] and reviewed in e.g. [11]. Applications of this idea in vibration control were investigated in [12].

The objectives of this research were to apply this method to the solution of differential equations, and time dependent problems in particular. The rest of this thesis is organized as follows. Chapter 2 describes preliminary theory and sets up the notation used throughout this thesis. Chapter 3 describes the actual work that was done. Section 3.1 applies the proposed method to a basic mass spring damper system, while Sections 3.2 and 3.3 describe developments for the one dimensional wave equation. Chapter 4 presents a summary, conclusions, and topics of future work.

Chapter 2

Theory

Contents

2.1 Notation

Vectors and matrices are bolded (e.g. $\mathbf{A}$, $\mathbf{v}$). Vector valued functions are bolded capital letters, for example $\mathbf{G}(\mathbf{x}$. The i'th component is written $G_i(\mathbf{x})$. To follow mechanical engineering conventions, x and $\mathbf{x}$ are reserved for spatial coordinates throughout this work. Generic inputs to a function are be denoted v and $\mathbf{v}$ for functions with scalar and vector inputs respectively. The bold $\mathbf{p}$ with subscripts is reserved for parameters of parametrized functions (Sec. 2.2). With an abuse of notation, it denotes a set of parameter vectors or matrices $\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}$. Time is denoted t. The superscript * is used to denote a desired value. For example, optimizing a set of parameters $\mathbf{p}$ to fit a loss function J will yield $\mathbf{p}*$. The exact solution to an equation in the unknown u is denoted u^*.

There are some concepts that occur frequently in this work and have letters reserved for them. The letter G is reserved for the boundary extension function. The letter D is reserved for the distance function. Both are introduced in Sec. 2.4 of this chapter. Their bolded equivalents $\mathbf{G}$, $\mathbf{D}$ are vector valued.

2.2 Parametrized function

First introduce the notion of a parametrized function. Strictly speaking, a function has all of its inputs and parameters as just inputs. A least square fit of the type $f(v) = av + b$ strictly speaking should be written $f = f(v, a, b)$. However, the function is optimized over a and b. The variables a, b are thus parameters. More generally, a vector valued function $\mathbf{F} = \mathbf{F}(\mathbf{v_1}, \mathbf{v_2}, \ldots, \mathbf{v_n}|\mathbf{p_1}, \mathbf{p_2}, \ldots \mathbf{p_m})$ maps the inputs $\mathbf{v_i}$ to an output. The parameters $\mathbf{p_{1,2,\ldots,n}}$ are modified to achieve the desired behavior of $\mathbf{F}(\mathbf{x}|\mathbf{p_{1,2,\ldots,n}})$. In general each of the parameters $\mathbf{p_i}$ is a matrix, but they can also be scalars. In the context of the least squares example, there is only one dimension. Thus, $\mathbf{F}(\mathbf{v}|\mathbf{p}) = F(v|a, b)$ with the two parameters $p_1 = a, p_2 = b$ optimized to achieve the desired behaviour of fitting a cloud of points.

Formally, achieving the desired behaviour of the function $\mathbf{f}(\{\mathbf{v_1}, \ldots, \mathbf{v_m}\}), \mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n}))$ can be described through minimization of a loss function J over the parameters

Formally define a parametrized function as

Definition 1 *A parametrized function* $\mathbf{F}(\mathbf{v}|\mathbf{p_1}, \mathbf{p_2}, \ldots, \mathbf{p_n})$ *is optimized over its parameters* $\mathbf{p_F}$ *to minimize some loss function* J

$$\mathbf{p_F}^* = \arg\min_{\mathbf{p_F}} J \tag{2.1}$$

to achieve some desired behaviour for the resultant optimized parametrized function $\mathbf{F}^*(\mathbf{v}) = \mathbf{F}(\mathbf{v}|\mathbf{p}^*)$.

In traditional applications, a set of known outputs $\boldsymbol{\gamma}_i$ and corresponding inputs $\mathbf{v}_i$ are

given and the goal is approximate the relationship between them. The loss function can then for example become the mean squared loss (MSE)

$$J_{\text{MSE}} = \sum_i (\boldsymbol{\gamma}_i - \Phi(\mathbf{x}_i))^2 \tag{2.2}$$

More complicated loss functions exist. In the present work, the goal is to approximate the solution to a differential equation using the neural network. Thus, the loss function is the error of the PDE to be solved.

2.3 Feedforward neural network

The feedforward neural network used in the present work consists of building blocks called layers with the l'th layer $\mathbf{f}^{(l)}$ mapping its input $\mathbf{x}^{(l)}$ to its output $\mathbf{h}^{(l)}$

$$\mathbf{f}^{(l)} \colon \mathbb{R}^{\text{in}} \to \mathbb{R}^{\text{out}} \tag{2.3}$$

$$\mathbf{x}^{(l)} \mapsto \sigma\left(\mathbf{W}^{(l)}\mathbf{x}^{(l)} + \mathbf{b}^{(l)}\right) \tag{2.4}$$

where $\mathbf{W}^{(l)}$ and $\mathbf{b}^{(l)}$ are the called the weights and biases at layer l. The nonlinear activation function $\sigma : \mathbb{R} \to \mathbb{R}$ is applied elementwise to its input. The feedforward network is a composition of layers. The output of each layer becomes the input to the next one i.e. $\mathbf{h}^{(l)} := \mathbf{x}_{l+1}$. Using L layers gives the neural network defined as

Definition 2 *Given appropriately sized weight matrices $\mathbf{W}_l$ and bias vectors $\mathbf{b}_l$ with $l = 1, 2, \ldots, L$, the feedforward neural network is defined as the parametrized function $\boldsymbol{\Phi}(\mathbf{v}|\mathbf{W_1}, \ldots, \mathbf{W_L}, \mathbf{b_1}, \ldots, \mathbf{b_L})$ that maps the input $\mathbf{v}$ as*

$$\boldsymbol{\Phi}(\mathbf{v}|\mathbf{W_1}, \ldots, \mathbf{W_L}, \mathbf{b_1}, \ldots, \mathbf{b_L}) = \mathbf{W_L}(\ldots \mathbf{W_2}(\sigma(\mathbf{W_1}\mathbf{h^{(0)}} + \mathbf{b_1})) + \mathbf{b_2}) + \mathbf{b_L} \tag{2.5}$$

with σ being an appropriately defined nonlinearity $\sigma : \mathbb{R} \to \mathbb{R}$ applied elementwise. The

weights and biases are collectively denoted as the parameters $\mathbf{p}_{\mathbf{\Phi}}$.

2.4 A simple example: The one dimensional heat equation

The one-dimensional heat equation is a simple boundary value problem (BVP) which will be used to illustrate the basic approach. It is given by the local equation

$$\frac{\partial^2 u}{\partial x^2} = h(x) \tag{2.6}$$

with $x \in [0, L]$ the spatial coordinate in a bar of length L, $u(x)$ is the temperature distribution at steady state, and a heat input distribution $h(x)$. Dirichlet boundary conditions will be considered and are written as

$$u(0) = u^*(0) \tag{2.7}$$

$$u(L) = u^*(L) \tag{2.8}$$

where $u^*(0)$ and $u^*(L)$ are used to denote the "desired" values of the the solution at the ends of the bar.

The goal is now to reformulate the problem defined by the Eq. (2.6) and boundary conditions (2.7)- (2.8) as an optimization problem of the type described in Def. 2.2. Form the loss function defined as the residual of Eq. (2.6), integrated over the domain:

$$J(\mathbf{W}, \mathbf{b}) = \int_0^L \left(\frac{\partial^2 u}{\partial x^2} - f(x) \right)^2 \mathrm{d}x \tag{2.9}$$

In simple cases, it might be possible to simplify this analytically. However in practice this is usually not possible. So approximate this integral as the sum of the integrand evaluated over

a set of N collocation points, divided by the number of points

$$J(\mathbf{W},\mathbf{b}) = \frac{1}{N}\sum_{x_i\in[0,L]}\left(\frac{\partial^2 u}{\partial x^2}\Big|_{x=x_i} - f(x_i)\right)^2 \tag{2.10}$$

The trouble with just minimizing the loss function in Eq. 2.10 is that it does not take into account the boundary conditions. There are two main ways to fix this. The first is to add the boundary conditions to the loss function. For Dirichlet BC's where $u(0) = u_0, u(L) = u_L$ this can take the form

$$J(\mathbf{W},\mathbf{b}) = \frac{1}{N}\sum_{x_i\in[0,L]}\left(\frac{\partial^2 u}{\partial x^2}\Big|_{x=x_i} - f(x_i)\right)^2 \mathrm{d}x + (u(0)-u_0)^2 + (u(L)-u_L)^2 \tag{2.11}$$

This is the approach taken in [13]. The second way is to design a solution ansatz that involves the neural network which by construction satisfies the boundary conditions. This is the approach taken in [9] and [8]. Introduce the boundary data extension function $G(x)$ and the smooth distance function $D(x)$. Now again use Def. (2) and set

$$u(x) := G(x) + D(x)\Phi(x) \tag{2.12}$$

$$x := \mathbf{h}^{(0)} \tag{2.13}$$

At the boundaries $x = 0, L$, the purpose of $G(x)$ is to satisfy the boundary conditions. The purpose of $D(x)$ is to ensure that the $D(x)\Phi(x)$ term does not affect the boundary conditions at all. For the heat equation with Dirichlet BCs this translates to $G(0) = u_0, G(L) = u_L$ and $D(0) = 0, D(L) = 0$. So set for example

$$G(x) = u_0 + \frac{u_L - u_0}{L}x \tag{2.14}$$

$$D(x) = x(L-x) \tag{2.15}$$

The approach used in the present work is the one described in Eqs. (2.12) and (2.13). In this

simple one-dimensional case, it is possible to form the distance and boundary extension functions analytically. In general, it can be necessary to obtain them by optimizing parametrized functions.

Chapter 3

Development for some basic structural dynamics equations

Contents

3.1 One variable: Mass Spring System

The one-dimensional mass-spring system is given by

$$m\ddot{u} + ku = h(t) \tag{3.1}$$

Initial conditions on displacement and velocity are prescribed as

$$u(0) = u^*(0) \tag{3.2}$$

$$u_t(0) = u_t^*(0) \tag{3.3}$$

In the context of Def. (1) set $v = t$ and $\mathbf{F}(v|\mathbf{p_F}) = f(t|\mathbf{p_f})$. The function $\mathbf{F}$ and its input $\mathbf{v}$ become the scalar valued f and v. The desired behaviour of $f(x|\mathbf{p_F})$ is to fit the local equation Eq. (3.1) and the initial conditions in Eqs. (3.2)- 3.3.

3.1.0.1 Solution ansatz: meeting initial conditions

The goal is to form a solution ansatz $f(t|\mathbf{p_f})$ that will meet the initial conditions Eqs. (3.2) through (3.3) for all $\mathbf{p_F}$. The parameters $\mathbf{p_F}$ will then be optimized to make $f(t|\mathbf{p_f})$ fit the local equation (3.1). Arbitrarily for now, introduce the boundary extension function $G(t)$ and the distance function $D(t)$ and form the solution ansatz for $u(t)$ as

$$f(t|\mathbf{p_F}) = u(t) = G(t) + D(t)\Phi(t|\mathbf{p}_\Phi) \tag{3.4}$$

Distance function

The distance function $D(t)$ is defined such that the product $D(t)\Phi(t|\mathbf{p}_\Phi)$ has no effect on initial conditions. In other words

$$D(0)\Phi(0) = 0 \tag{3.5}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}[D(t)\Phi(t)]|_{t=0} = 0 \tag{3.6}$$

There are infinitely many $D(t)$ that satisfy this condition. Any polynomial of the form $D(t) = \sum_{i=n}^{N} a_i t^n \ i \geq 2$ will work.

Boundary extension function

The role of the boundary extension function is only to satisfy the initial conditions. Indeed, if Eq. (3.6) is satisfied, then

$$u(0) = G(0) + D(0)\Phi(0) = G(0) \tag{3.7}$$

$$\frac{\mathrm{d}u}{\mathrm{d}t}|_{t=0} = \frac{\mathrm{d}}{\mathrm{d}t}G\big|_{t=0} + \frac{\mathrm{d}}{\mathrm{d}t}[D(t)\Phi(t)]\big|_{t=0} = \frac{\mathrm{d}}{\mathrm{d}t}G\big|_{t=0} \tag{3.8}$$

Which should satisfy the initial conditions. Therefore,

$$G(0) = u*(0) \tag{3.9}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}G\big|_{t=0} = u_t^*(0) \tag{3.10}$$

To satisfy the conditions in Eqs. (3.9) and (3.10), set

$$G(t) = u^*(0) + u_t^*(0)t \tag{3.11}$$

3.1.1 Residual minimizing function

The function $\Phi(t|\mathbf{p}_\Phi)$ is the part of the ansatz that, when optimized, actually minimizes the PDE error. It can be any continuous parametrized function with appropriately sized input and output dimensions. In this work, it is taken to be a shallow feedforward neural network as per Def. (2).

3.1.2 Final solution setup

Pick a set of collocation points $t_1, t_2, \ldots, t_i, \ldots, t_N$ on the domain of interest $[0, t_{\max}]$. Then,

$$f(t|\mathbf{p}_\mathbf{F}) = u(t) = G(t) + D(t)\Phi(t|\mathbf{p}_\Phi) \tag{3.12}$$

$$= u^*(0) + u_t^*(0)t + t^2\Phi(t|\mathbf{p}_\Phi) \tag{3.13}$$

with $\mathbf{p}_\mathbf{F} = \mathbf{p}_\Phi$ i.e. the only parameters left to optimize at this point are those associated with $\mathbf{p}_\Phi$. Then introduce the PDE residual, defined as the squared error of Eq. (3.1).

$$R(g(t), t) = (m\ddot{g}(t) + kg(t) - h(t))^2 \tag{3.14}$$

which is only zero if $f(t)$ solves the local equation exactly. The cost function becomes the integral of the PDE residual in Eq. (3.14). This integral is approximated as a Riemann sum over the collocation points.

$$J(\mathbf{p}_\mathbf{F}) = \frac{1}{N}\sum_{i=1}^{N} R(f(t|\mathbf{p}_\mathbf{F}), t_i) \tag{3.15}$$

3.1.3 Test case

Consider the unforced mass-spring system with $m = k = 1, h(t) = 0$ and initial conditions $u(0) = u^*(0) = 1$, $u_t(0) = u_t^*(0) = 0$. The exact solution is $u_{\text{exact}} = \cos(t)$ The solution ansatz is defined by Eq. (3.4) with $G(t)$ from Eq. (3.11) and $D(t) = \frac{0.3}{t_{\max}}t^2$. The results of

an example run with 20 equally spaced collocation points are presented in Figs. 3.2 and 3.3.

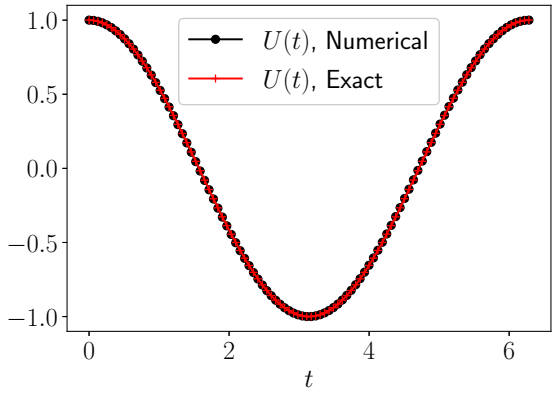


Figure 3.1: Comparison of obtained numerical and exact solutions

The optimization for this example was run using a limited memory LBFGS algorithm for 400

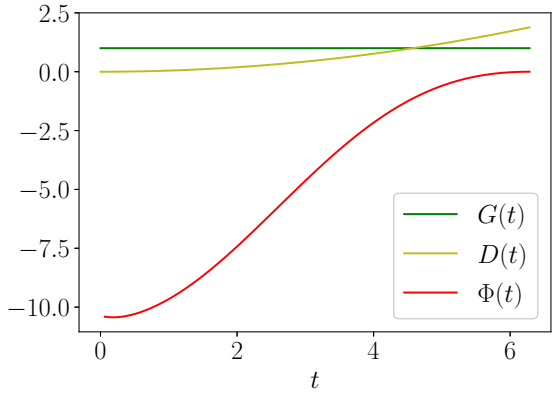


Figure 3.2: Components of numerical solution

function evaluations. The numerical solution residual is plotted in Fig. 3.3.

3.1.4 Convergence study

Convergence was considered with respect to the number of degrees of freedom (d.o.f.) in the neural network approximation for $\Phi(t|p_\Phi)$, as well as with respect to the number of points N_t

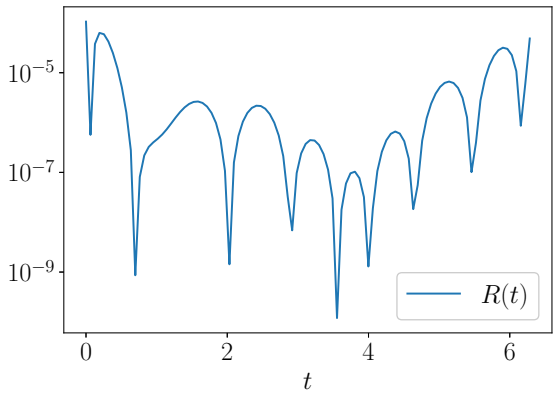


Figure 3.3: Residual of numerical solution for mass spring test case

used in approximating the residual average over the time domain of interest

$$\frac{1}{t_{\max}} \int_0^{t_{\max}} R(t, \mathbf{p_f}) \mathrm{d}t \approx \frac{1}{N_t} \sum_{i=1}^{N_t} R(t_i, \mathbf{p_f}) \tag{3.16}$$

The same parameter values were considered as for the test case in subsubsection 3.1.3. The mass m, spring constant k, and forcing function $h(t)$ were set to $m = 1$, $k = 1$, $h(t) = 0$ and the initial conditions were initial position $u_0 = 1$ and initial velocity $v_0 = 0$. The exact solution is $u_{\text{exact}}(t) = \cos(t)$. The error is obtained as $\int_0^{t_{\max}} (u_{\text{exact}}(x) - u_{\text{num}}(x))^2 \mathrm{d}t$ and is approximated as $\sum_{i=0}^{N_{\text{err}}} (u_{\text{exact}}(x_i) - u_{\text{num}}(x_i))^2$. The number of points for error approximation N_{err} is very large.

Feedforward neural networks have two hyperparameters that can be modified to increase degrees of freedom. Those hyperparameters are network width and depth. Thus, the same number of degrees of freedom can be obtained with different network configurations. To quantify convergence, the maximum number of function evaluations in the optimization algorithm was held fixed, while varying the network configuration. Convergence results with respect to the number of degrees of freedom are presented in Fig. 3.4. Convergence results with respect to number of points N_t in approximating the residual average are presented in Fig. 3.5. The convergence results with respect to number of degrees of freedom were

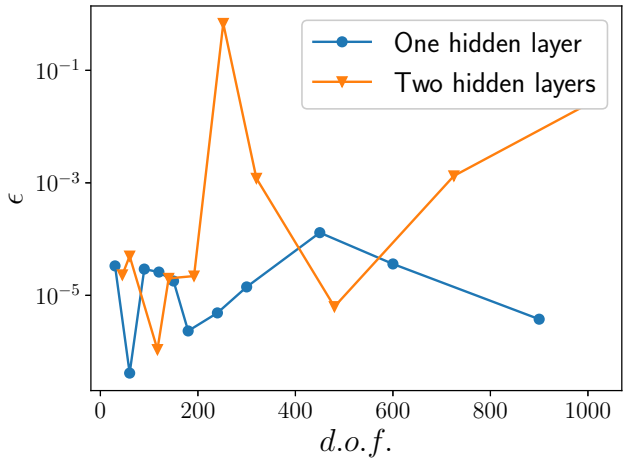


Figure 3.4: Error $\epsilon = \int_0^{t_{\max}} (u_{\text{exact}} - u_{\text{num}})^2 \mathrm{d}t$ with respect to number of degrees of freedom for different layer amounts

not convincing. A possible reason was the increase in the amount of variables to optimize. Although the complexity of the neural network increased with number of d.o.f., it was perhaps harder for the optimizer to find a good minimum. The shape of the convergence plot with respect to the amount of points used in the residual average approximation demonstrated a better downward trend, although it was still not as nice as convergence plots are expected to be.

3.2 Two variables: One dimensional wave equation

The one-dimensional (1D) wave equation is a PDE that can serve as a model for a longitudinally vibrating bar or a vibrating string. The wave equation in one dimension is written as

$$u_{tt} = c^2 u_{xx} \; [x, t] \in \Omega \tag{3.17}$$

where x being space and t time. The domain Ω is the rectangle $[0, L] \times [0, t_{\max}]$. Normally, the time and space dimensions are treated differently. For example, in finite elements, shape functions in space are used to form a system of ODEs in time. These ODEs are subsequently

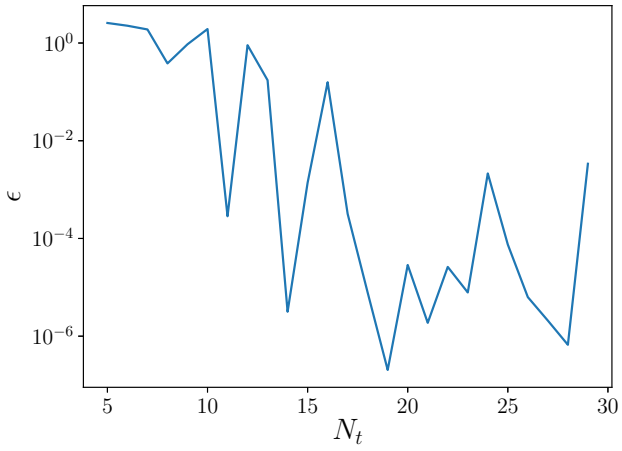


Figure 3.5: Error $\epsilon = \int_0^{t_{\max}} (u_{\text{exact}} - u_{\text{num}})^2 \mathrm{d}t$ with respect to number of points in residual average approximation

solved with a a time integrator (such as a Runge Kutta scheme). However, in this approach, space and time form a rectangular domain and the whole problem is solved "in one shot". Dirichlet boundary conditions will be examined. Thus the boundary and initial conditions are

$$u(x,0) = u^*(x,0) \tag{3.18}$$

$$u_t(x,0) = u_t^*(x,0) \tag{3.19}$$

$$u(0,t) = u^*(0,t) \tag{3.20}$$

$$u(L,t) = u^*(L,t) \tag{3.21}$$

The approach is almost the same as in Sec. 3.1 with a few differences. Instead of only time, everything is now a function of space x and time t.

$$u(x,t) = G(x,t) + D(x,t)\Phi(x,t) \tag{3.22}$$

where the boundary extension function $G(x,t)$ needs to satisfy

$$G(x,0) = u^*(x,0) \tag{3.23}$$

$$G_t(x,0) = u_t^*(x,0) \tag{3.24}$$

$$G(0,t) = u^*(0,t) \tag{3.25}$$

$$G(L,t) = u^*(L,t) \tag{3.26}$$

and the distance function $D(x,t)$ needs to satisfy

$$D(x,0) = 0 \tag{3.27}$$

$$D_t(x,0) = 0 \tag{3.28}$$

$$D(0,t) = 0 \tag{3.29}$$

$$D(L,t) = 0 \tag{3.30}$$

Set the boundary extension function $G(x,t)$ to depend on some parameters i.e. $G = G(x,t|\mathbf{p_G})$. The parameters $\mathbf{p_G}$ are found through

$$\mathbf{p_G}^* = \arg\min_{\mathbf{p_G}} J_G \tag{3.31}$$

$$= \arg\min_{\mathbf{p_G}} \int_0^L (G(x,0|\mathbf{p_G}) - u^*(x,0))^2 + (G_t(x,0|\mathbf{p_G}) - u_t^*(x,0))^2 \mathrm{d}x + \\ + \int_0^{t_{\max}} (G(0,t|\mathbf{p_G}) - u^*(0,t))^2 + (G(L,t|\mathbf{p_G}) - u^*(L,t))^2 \mathrm{d}t \tag{3.32}$$

Similarly

$$\mathbf{p_D}^* = \arg\min_{\mathbf{p_D}} J_D \tag{3.33}$$

$$= \arg\min_{\mathbf{p_D}} \int_0^L (D(x,0))^2 + (D_t(x,0))^2 \mathrm{d}x + \int_0^{t_{\max}} (D(0,t))^2 + (D(L,t))^2 \mathrm{d}t \tag{3.34}$$

where all the integrals are approximated as Riemann sums along the corresponding boundary.

3.2.1 Test case

The results for an example run of wave equation for initial conditions $u_0(x) = \sin(x)$ and $v_0(x) = 0$ and zero displacement Dirichlet boundary conditions are presented in Fig. 3.6. They are not encouraging, and it is at the time of writing not clear if the issues are caused by software issues or in the fundamental setup of the theory. A large residual spike is consistently observed near the domain boundary.

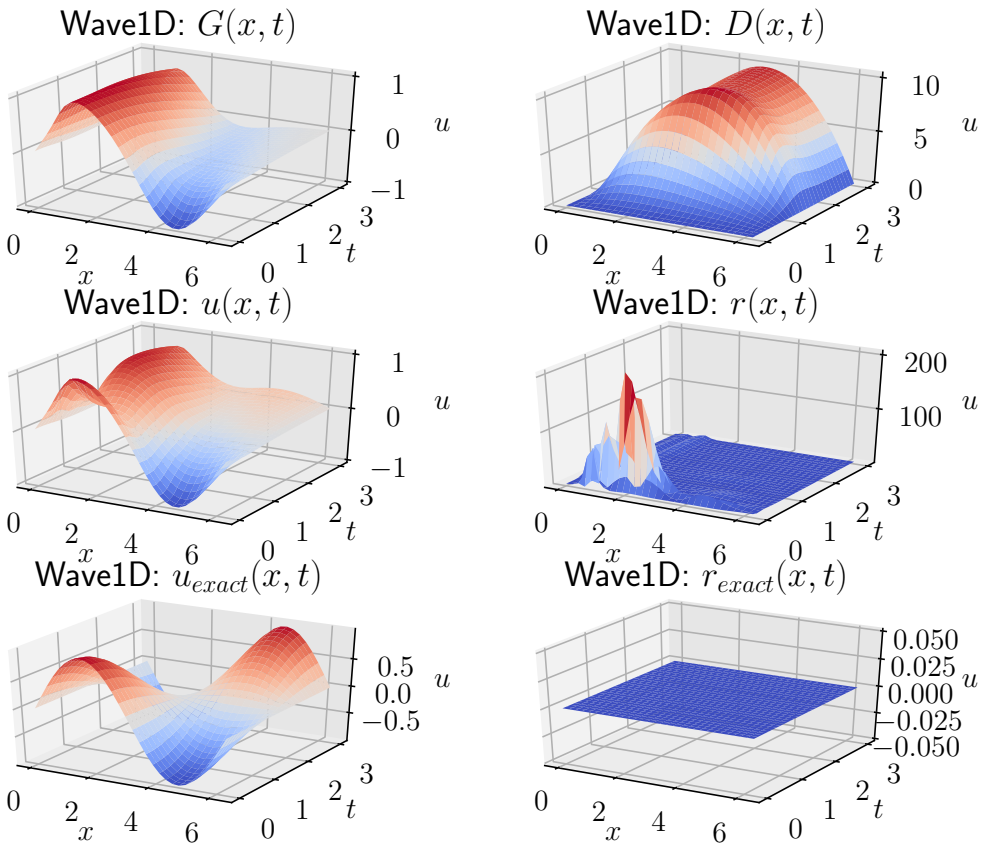


Figure 3.6: Results for a wave equation run with the residual fitting function $\Phi(x,t|\mathbf{p}_\Phi)$ being a neural network with three layers and 10 neurons each.

3.2.2 Note on choosing distance and boundary extension functions

In practice, some choices might work better than others for the optimization. For instance, picking a $D(t)$ which is discontinuous can make it difficult to find parameters that will satisfy the differential equation.

However, what makes the optimization work better is an opaque topic and is beyond the scope of this work.

3.3 Wave equation as a first order system

3.3.1 Notation

With an abuse of notation, within this section, u_t, u_x denote independent variables. Partial derivatives will be denoted in Leibniz notation. For example $\frac{\partial u}{\partial t}$ is the derivative of u with respect to t while u_t is a variable of its own. The reason for this will become clear shortly.

3.3.2 First order system formulation

The 1D wave equation (3.17) can be recast as first order system of three PDEs in the three unknowns $u(x,t), u_t(x,t)$ and $u_x(x,t)$. Eq. (3.17) takes the form

$$\frac{\partial}{\partial x}\begin{pmatrix} u(x,t) \\ u_t(x,t) \\ u_x(x,t) \end{pmatrix} - \frac{\partial}{\partial t}\begin{pmatrix} u(x,t) \\ u_x(x,t) \\ u_t(x,t) \end{pmatrix} = \begin{pmatrix} u_x(x,t) - u_t(x,t) \\ 0 \\ 0 \end{pmatrix} \quad [x,t] \in \Omega \tag{3.35}$$

Dirichlet boundary conditions will be examined. Thus specify the boundary and initial conditions as

$$u(x,0) = u^*(x,0) \tag{3.36}$$

$$u_t(x,0) = u_t^*(x,0) \tag{3.37}$$

$$u(0,t) = u^*(0,t) \tag{3.38}$$

$$u(L,t) = u^*(L,t) \tag{3.39}$$

It is worth again emphasizing that the unknowns $u(x,t), u_t(x,t), u_x(x,t)$ are now independent. The partial derivative $\frac{\partial u}{\partial t}(x,t)$ is only equal to $u_t(x,t)$ if the system of equations in Eq. (3.35) is solved exactly. This is also the reason why Eqs. (3.36)- (3.39) look the same as Eqs. (3.18)- (3.21) in Chap. 3 Sec. 3.2. In the single equation formulation u_t was the derivative w.r.t. t while here it is a variable of its own.

3.3.3 Setup of optimization problem

In the context of Def. (1), which defined parametrized functions, set

$$v_1, v_2 = x, t \tag{3.40}$$

$$\mathbf{F}(v_1, v_2|\mathbf{p_F}) = \mathbf{F}(x,t|\mathbf{p_F}) = \begin{pmatrix} u(x,t) \\ u_t(x,t) \\ u_x(x,t) \end{pmatrix} \tag{3.41}$$

The desired behaviour of $\mathbf{F}(x,t|\mathbf{p_F})$ is to fit the local equation (3.35) and the boundary conditions in Eqs. (3.36)- (3.39).

3.3.4 Solution ansatz: meeting the initial and boundary conditions

The solution ansatz $\mathbf{F}(\mathbf{x}, \mathbf{t}|\mathbf{p_F})$ for the first order wave equation formulation in Eq. 3.35 is defined similarly to the first order formulation. However, the boundary extension $\mathbf{G}(\mathbf{v})\colon \mathbb{R}^2 \to \mathbb{R}^3$ and distance $\mathbf{D}(\mathbf{v})\colon \mathbb{R}^2 \to \mathbb{R}^3$. They have a desired behaviour to be achieved by setting some parameters. Thus they are written $\mathbf{G}(\mathbf{v}|\mathbf{p_G})$ and $\mathbf{D}(\mathbf{v}|\mathbf{p_D})$. The solution ansatz parallels the second order formulation case in Eq. (3.41) and takes the form

$$\mathbf{F}(x,t|\mathbf{p_F}) = \begin{pmatrix} u(x,t) \\ u_t(x,t) \\ u_x(x,t) \end{pmatrix} = \mathbf{G}(x,t|\mathbf{p_G}) + \mathbf{D}(x,t|\mathbf{p_D}) \odot \mathbf{\Phi}(x,t|\mathbf{p_\Phi}) \tag{3.42}$$

where $\odot$ denotes elementwise multiplication, also known as the Hadamard product. The purpose of $\mathbf{G}(x,t|\mathbf{p_G})$ is to fit the boundary and initial conditions, while $\mathbf{D}(x,t|\mathbf{p_D})$ ensures that the product $\mathbf{D}(x,t|\mathbf{p_D}) \odot \mathbf{\Phi}(x,t|\mathbf{p_\Phi})$ does not affect the initial and boundary conditions whatever $\mathbf{\Phi}(x,t|\mathbf{p_\Phi}))$ is.

Boundary extension function

The distance function $\mathbf{G}(x,t|\mathbf{p_G})$ needs to fit the boundary and initial conditions in Eqs. 3.36-3.39. Thus require $G_1(\mathbf{0}, \mathbf{t}|\mathbf{p_G}) = u^*(0,t)$, $G_1(\mathbf{L}, \mathbf{t}|\mathbf{p_G}) = u^*(L,t)$, $G_1(\mathbf{x}, \mathbf{0}|\mathbf{p_G}) = u^*(x,0)$, and $G_2(\mathbf{x}, \mathbf{0}|\mathbf{p_G}) = u_t^*(x,0,t)$. The extension to Neumann boundary conditions is straightforward by imposing requirements on G_3. The boundary and initial conditions are met by forming the loss function

$$J_G = (G_1(0,t|\mathbf{p_G}) - u^*(0,t))^2 + (G_1(L,t|\mathbf{p_G}) - u^*(L,t))^2 + \tag{3.43}$$

$$+ (G_1(x,0|\mathbf{p_G}) - u^*(x,0))^2 + (G_2(x,0|\mathbf{p_G}) - u_t^*(x,0))^2 \tag{3.44}$$

and finding a "good enough" local minimum

$$\mathbf{p}_{\mathbf{G}}^{*} = \arg\min_{\mathbf{p}_{\mathbf{G}}} J_G \tag{3.45}$$

Distance function

The distance function $\mathbf{D}(x,t|\mathbf{p}_{\mathbf{D}})$ ensures the product $\mathbf{D}(x,t|\mathbf{p}_{\mathbf{D}}) \odot \mathbf{\Phi}(x,t|\mathbf{p}_{\mathbf{\Phi}})$ does not affect the boundary conditions. In other words, $\mathbf{D}(x,t|\mathbf{p}_{\mathbf{D}})$ needs to be zero wherever the boundary conditions are specified. This is met by forming the loss function

$$J_D = (D_1(0,t|\mathbf{p}_{\mathbf{G}}))^2 + (D_1(L,t|\mathbf{p}_{\mathbf{G}}))^2 + (D_1(x,0|\mathbf{p}_{\mathbf{G}}))^2 + (D_2(x,0|\mathbf{p}_{\mathbf{G}}))^2 \tag{3.46}$$

and finding a "good enough" local minimum to obtain

$$\mathbf{p}_{\mathbf{D}}^{*} = \arg\min_{\mathbf{p}_{\mathbf{D}}} J_D \tag{3.47}$$

3.3.5 Optimizing to fit the local equation

After finding p_G^*, p_D^*, the expression

$$\mathbf{F}(x,t|\mathbf{p}_{\mathbf{\Phi}}) = \begin{pmatrix} u(x,t) \\ u_t(x,t) \\ u_x(x,t) \end{pmatrix} = \mathbf{G}^*(x,t|\mathbf{p}_{\mathbf{G}}^*) + \mathbf{D}^*(x,t|\mathbf{p}_{\mathbf{D}}^*) \odot \mathbf{\Phi}(x,t|\mathbf{p}_{\mathbf{\Phi}}) \tag{3.48}$$

satisfies the initial and boundary condition. This leaves the local equation in Eq. (3.35) to be solved by optimizing the parameters $\mathbf{p}_{\mathbf{\Phi}}$ to minimize the local equation error integrated over the domain Ω. This error is obtained by putting all the terms in Eq. (3.35) on one side,

taking the vector norm, and integrating over Ω.

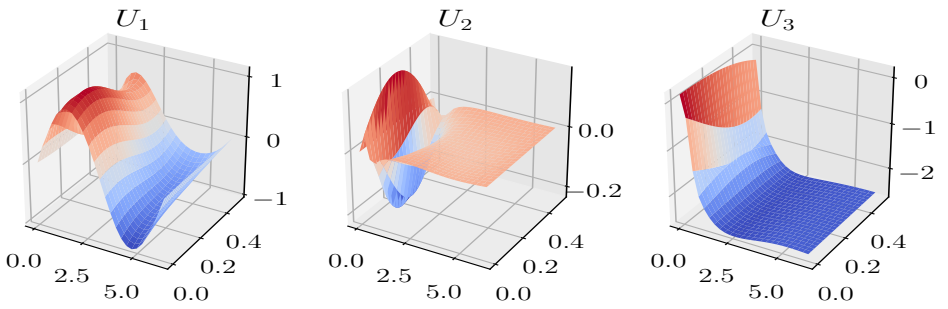
$$J_{local}(\mathbf{p_\Phi}) = \int_\Omega \left\| \frac{\partial}{\partial x} \begin{pmatrix} u(x,t) \\ u_t(x,t) \\ u_x(x,t) \end{pmatrix} - \frac{\partial}{\partial t} \begin{pmatrix} u(x,t) \\ u_t(x,t) \\ u_x(x,t) \end{pmatrix} - \begin{pmatrix} u_t(x,t) - u_x(x,t) \\ 0 \\ 0 \end{pmatrix} \right\|^2 \mathrm{d}t\mathrm{d}x \quad (3.49)$$

The integral is approximated as a Riemann sum over a set of collocation points. It is worth emphasizing that $J(\mathbf{p_\Phi})$ does not depend on either the variables x, t or the unknowns u, u_t, u_x. Once the parameters $\mathbf{p_F}$ are set, the partial derivatives of $\mathbf{F}(\mathbf{x}, \mathbf{t}|\mathbf{p_F})$ are known and set $J(\mathbf{p_F})$. The final solution is

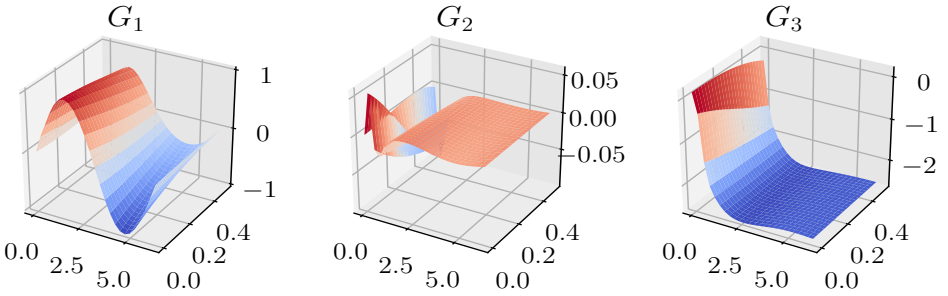
$$\mathbf{F}^*(x,t|\mathbf{p}^*_\Phi) = \begin{pmatrix} u(x,t) \\ u_t(x,t) \\ u_x(x,t) \end{pmatrix} = \mathbf{G}^*(x,t|\mathbf{p}^*_\mathbf{G}) + \mathbf{D}^*(x,t|\mathbf{p}^*_\mathbf{D}) \odot \mathbf{\Phi}(x,t|\mathbf{p_\Phi}) \quad (3.50)$$

3.3.6 Test case

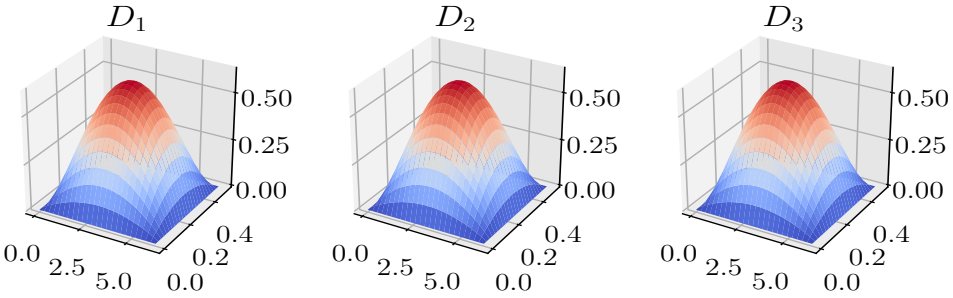
The results for an optimization run are shown in Fig. 3.7c. The first through last elements of U, D, G correspond to the $u(x,t), u_t(x,t), u_x(x,t)$ respectively.



(a) Plots of the elements of U



(b) Plots of the elements of D



(c) Plots of the elements of G

Figure 3.7: Example solution for wave equation recast in first order form. The three components U_1, U_2 and U_3 correspond to the displacement u, its time derivative u_t and its space derivative u_x respectively.

Chapter 4

Conclusion

A least squares approach to solving differential equations was presented and applied to the mass spring and one-dimensional wave equations. The method worked well for the mass spring system test case but failed for the more complicated wave equation. In the case of the wave equation, a first order system formulation was investigated in an attempt to reduce the method's computational cost. The results were unsatisfactory, similar to the second order formulation. It is unclear whether the wave equation failure was due to an error in the author's implementation or a property of the optimization problem setup.

Future work includes resolution of issues with the wave equation, investigation of different forms for the solution ansatz, and convergence studies with respect to hyperparameters of the neural network used.

Appendix A

Replicating the results

Code to replicate these results is located at `https://github.com/vkorotkine/Honours_Thesis`

Bibliography

[1] J. S. Hesthaven and T. Warburton. *Nodal discontinuous Galerkin methods: algorithms, analysis, and applications*. Springer Science & Business Media, 2007. DOI: 10.1007/978-0-387-72067-8.

[2] A. Idesman. "Optimal reduction of numerical dispersion for wave propagation problems. Part 1: Application to 1-D isogeometric elements". *Computer Methods in Applied Mechanics and Engineering* 317 (2017), pp. 970–992. DOI: 10.1016/j.cma.2017.01.014.

[3] D. P. Karadimou and N.-C. Markatos. "Study of the Numerical Diffusion in Computational Calculations". *Numerical Simulations in Engineering and Science* (2018), p. 65. DOI: 10.5772/intechopen.75660.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Nets". *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. Curran Associates, Inc., 2014, pp. 2672–2680. URL: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

[5] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Vol. 1. 10. Springer series in statistics New York, 2001. URL: https://web.stanford.edu/~hastie/Papers/ESLII.pdf.

[6] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber. "LSTM: A search space odyssey". *IEEE transactions on neural networks and learning systems* 28.10 (2016), pp. 2222–2232. DOI: 10.1109/TNNLS.2016.2582924.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[8] I. E. Lagaris, A. Likas, and D. I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". *IEEE Transactions on Neural Networks* 9.5 (1998), pp. 987–1000. DOI: 10.1109/72.712178.

[9] J. Berg and K. Nyström. "A unified deep artificial neural network approach to partial differential equations in complex geometries". *Neurocomputing* 317 (2018), pp. 28–41. ISSN: 0925-2312. DOI: 10.1016/j.neucom.2018.06.056.

[10] A. Malek and R. S. Beidokhti. "Numerical solution for high order differential equations using a hybrid neural network-optimization method". *Applied Mathematics and Computation* 183.1 (2006), pp. 260–271. DOI: 10.1016/j.amc.2006.05.068.

[11] M. Kumar and N. Yadav. "Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey". *Computers & Mathematics with Applications* 62.10 (2011), pp. 3796–3811. DOI: 10.1016/j.camwa.2011.09.028.

[12] H. Alli, A. Ucar, and Y. Demir. "The solutions of vibration control problems using artificial neural networks". *Journal of the Franklin Institute* 340.5 (2003), pp. 307–325. DOI: 10.1016/S0016-0032(03)00036-X.

[13] J. Sirignano and K. Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations". *Journal of Computational Physics* 375 (2018), pp. 1339–1364. DOI: 10.1016/j.jcp.2018.08.029.