UNIVERSIDADE ESTADUAL DE CAMPINAS
Faculdade de Engenharia Mecânica

Carlos Jose Gonzalez Rojas

Parameter identification for a damage model using a physics informed neural network

Identificação de parâmetros de um modelo de dano usando uma rede neural informada por leis físicas

CAMPINAS
2020

Carlos Jose Gonzalez Rojas

Parameter identification for a damage model using a physics informed neural network

Identificação de parâmetros de um modelo de dano usando uma rede neural informada por leis físicas

Dissertação apresentada á Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Mecânica na área de Mecânica dos Sólidos e Projeto Mecânico.

Dissertation presented to the School of Mechanical Engineering of the University of Campinas in partial fulfillment of the requirements for the degree of Master in Mechanical Engineering, in the area of Solid Mechanics and Mechanical Design.

Orientador : Marco Lúcio Bittencourt

Coorientador : José Luiz Boldrini

ESTE EXEMPLAR CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DEFENDIDA PELO ALUNO CARLOS JOSE GONZALEZ ROJAS E ORIENTADA PELO PROF. DR. MARCO LÚCIO BITTENCOURT.

CAMPINAS

2020

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Luciana Pietrosanto Milla - CRB 8/8129

G589p

Gonzalez Rojas, Carlos Jose, 1993-
Parameter identification for a damage model using a physics informed neural network / Carlos Jose Gonzalez Rojas. – Campinas, SP : [s.n.], 2020.

Orientador: Marco Lúcio Bittencourt.
Coorientador: José Luiz Boldrini.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Redes neurais (Computação). 2. Identificação de sistemas. 3. Mecânica do dano contínuo. I. Bittencourt, Marco Lúcio, 1964-. II. Boldrini, José Luiz, 1952-. III. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Identificação de parâmetros de um modelo de dano usando uma rede neural informada por leis físicas
Palavras-chave em inglês:
Neural networks (Computer science)
System identification
Continuum damage mechanics
Área de concentração: Mecânica dos Sólidos e Projeto Mecânico
Titulação: Mestre em Engenharia Mecânica
Banca examinadora:
Marco Lúcio Bittencourt [Orientador]
Josué Labaki Silva
Larissa Driemeier
Data de defesa: 17-04-2020
Programa de Pós-Graduação: Engenharia Mecânica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: 0000-0001-7750-9559
- Currículo Lattes do autor: http://lattes.cnpq.br/4116454059553792

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

DISSERTAÇÃO DE MESTRADO ACADÊMICO

Parameter identification for a damage model using a physics informed neural network

Identificação de parâmetros de um modelo de dano usando uma rede neural informada por leis físicas

Autor: Carlos Jose Gonzalez Rojas
Orientador: Prof. Dr. Marco Lúcio Bittencourt
Coorientador: Prof. Dr. José Luiz Boldrini

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação:

Prof. Dr. Marco Lúcio Bittencourt
FEM/UNICAMP

Prof. Dr. Josué Labaki Silva
FEM/UNICAMP

Prof. Dra. Larissa Driemeier
PMR/EPUSP

A Ata de Defesa com as respectivas assinaturas dos membros encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 17 de Abril de 2020.

Acknowledgements

Eu gostaria de agradecer a todas as pessoas que direta ou indiretamente fizeram parte deste processo.

A minha avó, minha mãe e minha irmã por seu infinito amor.

Ao professor Dr. Marco Lúcio Bittencourt pelo apoio, dedicação e confiança.

Ao professor Dr. José Luiz Boldrini pelas suas valiosas ideias e seus aportes no desenvolvimento deste trabalho.

Aos meus colegas do lab pelo bom ambiente de trabalho. Em especial agradeço ao Geovane, o Matheus, a Thais e o Renan, pela sua amizade.

A Ximena por todo seu amor, carinho e compreensão.

A UNICAMP e a FEM pela oportunidade de realizar este trabalho.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Código de Financiamento 001.

Resumo

Este trabalho aplica conceitos de redes neurais artificiais para identificar os parâmetros de um modelo matemático baseado em campos de fase para dano e fratura. A mecânica do dano é a parte da mecânica do contínuo que modela os efeitos da formação de micro-defeitos usando variáveis de estado no nível macroscópico. As equações que definem o modelo são derivadas de leis fundamentais da física e fornecem relações importantes entre as variáveis de estado. Simulações utilizando o modelo considerado neste trabalho produzem bons resultados qualitativos e quantitativos, mas muitos parâmetros devem ser ajustados para reproduzir um determinado comportamento material. Considera-se a identificação dos parâmetros do modelo por meio da resolução de um problema inverso que emprega dados pseudo-experimentais para encontrar valores que produzem o melhor ajuste aos dados. Aplica-se uma rede neural informada por leis físicas e são combinados alguns métodos clássicos de estimativa para identificar os parâmetros materiais que aparecem na equação de dano do modelo. A estratégia aplicada é composta por uma rede neural que atua como uma função aproximadora da evolução do dano com sua saída regularizada utilizando o resíduo da equação diferencial. Três estágios de otimização buscam os melhores valores possíveis para os parâmetros materiais e da rede neural. O treinamento alterna entre o ajuste de apenas os dados pseudo-experimentais ou da perda total que inclui os termos de regularização. A robustez do método na presença de ruído nos dados de treinamento e a capacidade de generalização desta metodologia são testadas usando um caso físico simples para o modelo de dano. Este procedimento lida melhor com a presença de ruído em comparação com um método de otimização restrito para a equação diferencial e também fornece boas aproximações dos parâmetros materiais e a evolução do dano.

Palavras-chave: Identificação de parâmetros, Modelo de dano, Redes neurais artificiais, Rede neural informada por leis físicas.

Abstract

This work applies concepts of artificial neural networks to identify the parameters of a mathematical model based on phase fields for damage and fracture. Damage mechanics is the part of the continuum mechanics that models the effects of micro-defect formation using state variables at the macroscopic level. The equations that define the model are derived from fundamental laws of physics and provide important relationships between state variables. Simulations using the model considered in this work produce good qualitative and quantitative results, but many parameters must be adjusted to reproduce a certain material behavior. The identification of model parameters is considered by solving an inverse problem that uses pseudo-experimental data to find the values that produce the best fit to the data. We apply a physics informed neural network and combine some classical estimation methods to identify the material parameters that appear in the damage equation of the model. Our strategy consists of a neural network that acts as an approximating function of the damage evolution with its output regularized using the residue of the differential equation. Three stages of optimization seek the best possible values for the neural network and the material parameters. The training alternates between the fitting of only the pseudo-experimental data or the total loss that includes the regularizing terms. We test the robustness of the method to noisy data and its generalization capabilities using a simple physical case for the damage model. This procedure deals better with noisy data in comparison with a PDE-constrained optimization method, and it also provides good approximations of the material parameters and the evolution of damage.

Keywords: Parameter identification, Damage model, Neural networks, Physics informed neural network.

List of Figures

List of Tables

Table of Contents

1 Introduction

Many mathematical models have been developed to describe the mechanical effects of progressive micro-crack formation using continuum damage modeling (HAKIM; KARMA, 2009; MIEHE; HOFACKER; WELSCHINGER, 2010; BORDEN et al., 2014). In order to represent this complex phenomenon, different approaches can be followed, but the framework and assumptions taken can restrict the generality and applicability of the models. Nowadays, different deterministic approaches have been applied to model damage and fatigue in materials, but few works have followed a thermodynamically consistent framework. Boldrini et al. (2016) addressed some mathematical deficiencies that have not been considered in previous works. Their model applies the phase field methodology to avoid limitations when dealing with crack initiation or branching and is based on the use of the basic principles of continuum mechanics. Although this model has achieved good qualitative and quantitative results, one of its difficulties is the appearance of some materials parameters, with not necessarily a clear physical meaning, that needs to be found in order to reproduce a particular material behavior.

The estimation of material parameters is intrinsically an inverse problem. According to Ghaboussi (2010), inverse problems are categorized into two classes. In the first class, the model and the output of the system are known and the objective is to find the input causing that response. In the second class, the input and the output of the model are defined and the aim is to find the system model (or the parameters that define it). This last class of analysis is called system identification and is the type of inverse problem that needs to be solved in this research work. As stated by Vogel (2002), there is a large mathematical literature on inverse problems divided into deterministic and non-deterministic approaches. In non-deterministic methodologies, the treatment of statistical aspects is prominent to fit a model using a set of data. On the other hand, in deterministic approaches, the methods disregard the intrinsic uncertainty in the model or the data used.

In this work, we treat the parameter identification problem using a deterministic approach. The identification is based on the construction of an objective function that measures differences between pseudo-experimental data and results obtained using a neural network. Though few consistent mathematical theories explain the suitability of neural networks for inverse problems, many applications have achieved good results even for ill-posed problems (ADLER; ÖKTEM, 2017; SEO et al., 2019; LI et al., 2020). We formulate an optimization problem where the minimization of an objective function leads to optimal parameters that reproduce the principles of a physics model. Instead of using costly approaches, such as the computation of forward solutions for each optimization step, we propose here the use of physics informed neural networks. With this alternative method, the solution is approximated using artificial neural networks and the parameters are estimated from the residue of a partial differential equation.

1.1 Literature review

As presented previously, this dissertation aims to solve a model parameter identification applying neural networks. However, the methodology applied in this work is concerned with the use of neural networks to solve differential equations as part of the identification strategy. Hence, the literature review starts with some articles that explored that topic and later describes contributions more related to the identification problem.

In one of the first studies in this area, Meade and Fernandez (1994) presented a method for the solution of ordinary differential equations with an interesting interpretation of the function approximation capability of feedforward neural networks. The authors took advantage of some similarities with basis expansions to constrain and assign certain roles to the inputs and parameters of a shallow network. In addition, the method of weighted residuals was applied to determine the output weights of the network and the problem was reduced to the solution of a system of algebraic equations. They applied a non-iterative approach to solving two simple problems and showed the accuracy of the approximation. Later, Lagaris, Likas and Fotiadis (1998) exploited the approximation capabilities of a neural network instead of transforming it into an explicit basis expansion. The authors introduced the residue of a differential equation into the learning process and removed the constraints of the objective function, adopting a mathematical expression that automatically satisfied the initial or boundary conditions. An algorithm minimized the loss function of the neural network and, in consequence, provided the learned parameters that allow an accurate approximation using the solution proposed. Examples of ordinary and partial differential equations were solved with good accuracy and some generalization advantages were observed when the proposed method was compared with the interpolation results of a finite element solution.

The ideas developed in Meade and Fernandez (1994) and Lagaris, Likas and Fotiadis (1998) inspired many other works and found applications in fields including quantum mechanics (LAGARIS; LIKAS; FOTIADIS, 1997; MANZHOS; CARRINGTON, 2009), simulation of biodegradation process (VINOD; KUMAR; REDDY, 2009), atomic and molecular physics (CAETANO et al., 2010), thin plate bending problems (LI et al., 2013), movement of contaminants in the subsurface (YADAV; YADAV; KIM, 2016), and different studies of control systems (HE; REIF; UNBEHAUEN, 2000; ALLI; UÇAR; DEMIR, 2003; EFFATI; PAKDAMAN, 2010; MASMOUDI et al., 2011). Those works also promoted other fronts of research concerned with the construction of hybrid methods (SMAOUI; AL-ENEZI, 2004; MALEK; BEIDOKHTI, 2006; TSOULOS; GAVRILIS; GLAVAS, 2009; DUA, 2011), improvement of optimization algorithms (AARTS; VEER, 2001; RUDD; FERRARI, 2015; BERG; NYSTRöM, 2018), quantification of errors (FOJDL; BRAUSE, 2008; FILICI, 2010; GROHS et al., 2019) and other important

contributions (MALL; CHAKRAVERTY, 2013; MALL; CHAKRAVERTY, 2014; MALL; CHAKRAVERTY, 2015; MALL; CHAKRAVERTY, 2016).

The review presented in the last paragraphs gives the basic support for the solution of differential equations through neural networks used in this dissertation. Before presenting the application of neural networks in the area of parameter estimation, it is important to introduce some concepts and methods regarding this problem. Parameter estimation is an important step in the development of accurate models on natural sciences, physics, engineering, and many other disciplines. Mathematical models developed to approximate dynamic processes often involve differential equations with unknown parameters. In a parameter identification problem, a measure of error is minimized to fit the model prediction with observed data. As described in Mehrkanoon, Falck and Suykens (2012), there are two main categories of methods used to estimate the parameters in the governing equation of a model. In the first category of methods, the differential equations are solved adopting random initial values for the parameters and their predictions are compared with experimental data. An objective function is defined to quantify the difference between the expected and obtained results and the model parameters are updated applying an optimization procedure. According to Moles (2003), this approach requires a high computational cost and almost 90% of the computation time is required to solve the model equations.The second category of methods substitute the solution of the governing equations adopting a functional approximation. Using this approximation, the required derivatives are calculated and the residue of the differential equation is constructed. Subsequently, this residue is minimized by adopting an optimization algorithm and the model parameters are estimated (MEHRKANOON; FALCK; SUYKENS, 2012).

Dua (2011) proposed a novel methodology for parameter estimation with the introduction of artificial neural networks as model approximators. This work is part of the second category presented in the last paragraph and can be defined as a decomposition algorithm (DUA; DUA, 2011). The author divides the problem into two steps, first an artificial neural network approximates the model after a process of training with measured data, and then the residue of the differential equation is defined through derivatives of the neural network. The parameters are part of the expression defining the residue and estimated employing an optimization procedure. The author presents some examples considering the estimation of parameters for kinetic models. Most of the examples are from models of chemical reactions and are given by systems of ordinary differential equations. In a subsequent work, the ideas described in Dua (2011) research were generalized. Dua and Dua (2011) introduced a simultaneous approach where the solution of ordinary differential equations and determination of the model parameters is performed at the same time. The objective function of the optimization process for this methodology is composed of a term that relates the differences between the predictions and observed data and the residue to satisfy the governing equations of the model. The method showed good results for different

examples solved with only one hidden layer and a small number of nodes.

More recently, Raissi, Perdikaris and Karniadakis (2017b) proposed a data-driven discovery algorithm for estimation of parameters in partial differential equations. Their methodology is called physics informed neural networks (PINNs) and the main differences with the work of Dua and Dua (2011) is the possibility to include boundary and initial conditions in the loss function of the neural network and the adoption of automatic differentiation to compute the derivatives of the network. These physics informed neural networks (PINNs) were applied in benchmark problems using a relatively small amount of data (system's solutions) and regularizing the system with the physics laws represented by differential equations. Similarly, in Tartakovsky et al. (2018), a PINN was applied to approximate the space-dependent coefficient in a linear diffusion equation and the constitutive relationship in a non-linear diffusion equation. Other successful applications can be found in Raissi, Perdikaris and Karniadakis (2018), Raissi, Yazdani and Karniadakis (2018), Raissi, Ramezani and Seshaiyer (2019), Tartakovsky, Barajas-Solano and He (2019), Tipireddy et al. (2019), Meng and Karniadakis (2020), where the inclusion of differential equations or constitutive equations as part of the loss function in neural networks have demonstrated to be efficient, accurate and suggest great promise for future applications.

This methodology has also inspired new flexible approaches where even the differential operators of the models are estimated from data. Long et al. (2017) proposed a novel method where the differential and nonlinear operators of a governing equation are learned without the definition of a fixed equation. The use of a feedforward neural network, called PDE-net, allowed the discovery of a hidden model using observational data and was also able to predict its dynamical behavior. Some examples using convection-diffusion equations uncovered the hidden equations from simulated data and provided good approximations for the dynamic behavior. Subsequently, in Rudy et al. (2019), a general method to identify the governing equations of a given model was proposed. This work is called a PDE-FIND data-driven model and the terms of the governing equations are selected from a library with linear, nonlinear, time and space differential operators. The method was tested in the identification of four canonical models and produced accurate approximations.

The starting point for the parameter estimation performed in this dissertation is the work developed by Raissi, Perdikaris and Karniadakis (2017b). We apply their methodology to identify the material parameters of the damage equation considering that the differential operators were derived with mathematical consistency following the basic principles of continuum mechanics. In addition, we consider that the parameters of the model are identifiable and that the neural network approximation is suitable in the case of ill-posed problems.

1.2 Motivation

The identification of material parameters is essentially an inverse problem where the output of a model is known and the aim is to find the parameters that produce these results. Parameter identification problems are common in different disciplines and play a significant role in the calibration and validation of physics models. Some of the difficulties of these type of inverse analyses are their dependence on expensive computations of forward solutions for the optimization process and the existence of non-unique solutions. On the other hand, application of neural networks to inverse problems has demonstrated many advantages in the reduction of computational cost and the solution of ill-posed problems.

1.3 Objectives

The main purpose of this work is the solution of a parameter identification problem for a damage model using an alternative approach that employs neural networks and enforces physics laws.

The specific objectives are listed as follows:

1. Consider a basic physical case for the application of the damage model;
2. Generate pseudo-experimental data implementing a numerical solution of the governing equations for the damage model;
3. Explore the use of neural networks in the solution of material estimation problems;
4. Implement a physics informed neural network to identify the material parameters of the damage equation.

1.4 Outline

This work is divided into 5 chapters. In this chapter, the introduction, followed by a literature review, motivation, and objectives were presented. Chapter 2 describes how the parameter identification can be formulated as an optimization problem, summarizes some classical estimation techniques and presents an overview of the damage model and the numerical

methods applied in the forward solution of their governing equations. Chapter 3 provides some fundamental concepts for neural networks and explains the physics informed methodology. In chapter 4 the estimation strategy proposed is detailed and the robustness and generalization capabilities of the method are evaluated. Finally, chapter 5 presents the conclusions and some suggestions for future research.

2 Parameter Identification of a Damage Model

The forward problem of solving the governing equations of a model has been extensively studied in mathematics and applied in engineering fields. However, to obtain realistic results from the solution of governing equations, physics models have to be validated and calibrated using experimental data. The process of validation and calibration requires the solution of an inverse input-output mapping and it is necessary to find the causes that lead to that state (BULJAK, 2012).

Partial differential identification problems are a class of inverse problems where the unknown inputs of a model are parameters entering into the governing equations. Identification problems arise in fields such as geophysics, fluids, structural mechanics, electromagnetics, biomedical and thermal sciences and there has been a steady interest in developing efficient estimation approaches for different applications.

This chapter presents a brief description of parameter identification in differential equations and introduces the damage model considered in the inverse analysis proposed.

2.1 Parameter identification in differential equations

As previously introduced, an important step in the formulation of a model is the connection of its governing equations with the experimental data that comes from observations. A general mathematical form used to represent several phenomena is

$$\mathscr{F}(x_1, ..., x_j, u, \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_j}, ..., \frac{\partial^2 u}{\partial x_1 \partial x_j}, \lambda) = 0, \tag{2.1}$$

where $u(x)$ is the state variable, x are the input variables $(x_1, ..., x_j)^T$ and λ is the vector of parameters $(\lambda_1, ..., \lambda_k)^T$. These parameters can be values defining a physical entity directly or coefficients in relationships that describe a physical process (ASTER BRIAN BORCHERS, 2019; FRASSO; JAEGER; LAMBERT, 2016).

The data collected from experimental observations is represented by

$$\hat{u}(x) = u(x) + \epsilon \tag{2.2}$$

where ϵ is assumed to be an independent normally distributed measurement error with mean zero and constant variance.

The objective of parameter identification in a model described by Eq. (2.1) is to find a set of parameter estimates $\hat{\lambda}$, that leads to minimal differences between the observed data $\hat{u}(x)$ and the solution of the differential equation $u(x, \hat{\lambda})$. Some important issues that need to be considered in this type of problem are the model suitability to represent the experimental data and if it is possible to uniquely identify the parameters with the observations available. As presented in Jadamba, Khan and Sama (2011), inverse problems are frequently ill-posed in the sense of Hadamard. Some of the causes of ill-conditioning are an insufficient approximation of models, data affected with noise and the lack of additional constraints.

The methods to estimate parameters in differential equations are divided into two major categories. The first category groups deterministic approaches where it is assumed that the state variable of the model is completely defined by its parameters, boundary conditions, and initial conditions. In this case, possible disturbances that can arise in the mathematical formulation of the model, the approximate solution of the governing equations and the observations are not directly accounted for in the estimation of the parameters. Conversely, in the second category, the use of stochastic methods systematically includes these uncertainties using frequentist or Bayesian approaches (VARZIRI; MCAULEY; MCLELLAN, 2008).

Several methods to estimate parameters in models described by ordinary or partial differential equations (ODEs or PDEs) have been developed through years (MüLLER; TIMMER, 2004; RAMSAY et al., 2007; VARZIRI; MCAULEY; MCLELLAN, 2008; CAO; HUANG; WU, 2012; XUN et al., 2013; FRASSO; JAEGER; LAMBERT, 2016) . Despite their different degrees of complexity, methods initially developed for identification of parameters in ODEs have been adapted and applied for models described by PDEs.

The first step common in any estimation method is the selection of the criterion to fit the data. A prominent criterion in inverse problems is the least square method which is adequate in cases where the uncertainties can be modeled using Gaussian distributions. This is the preferred standard because it simplifies the calculations in the optimization process but has difficulties dealing with outliers in the data. An alternative to avoid that problem is the least absolute value criterion which reduces the sensitivity to errors introduced by unlikely observations (TARANTOLA, 2005). After the selection of fitting criteria, the next step is to pose the identification of parameters as an optimization problem. The first option is to propose the constrained optimization presented in the following expression:

$$\begin{aligned} \min_{\lambda} \quad & \sum_{i=1}^{N} (u(x) - \hat{u}(x))^2 \\ \text{s.t.} \quad & \mathscr{F}(x_1, ..., x_j, u, \frac{\partial u}{\partial x_1}, ..., \frac{\partial u}{\partial x_j}, ..., \frac{\partial^2 u}{\partial x_1 \partial x_j}, \lambda) = 0, \end{aligned} \tag{2.3}$$

where the objective is to fit the experimental data $\hat{u}(x)$ to the state variable of the model $u(x)$ using a least square criterion and with the differential equation as the constraint. In many situations governing equations cannot be solved analytically, so, in addition to the data fitting procedure, it is necessary to introduce a numerical method to approximate the solution of the model equations and the sensitivity of the state variable to the parameters of the model. This increases the computational complexity of the problem and requires the implementation of parallel optimized low-level code and high-level abstraction frameworks for automatic differentiation that only some software libraries such as PETSc (BALAY et al., 2019), DOpElib (GOLL; WICK; WOLLNER, 2017) and dolfin-adjoint (FUNKE; FARRELL, 2013) have developed.

A second option, widely implemented in identification problems, is the use of function basis expansions to approximate the output of the model. This approach was introduced in the work of Varah (1982) and its main advantage is the decrease in the computational cost of directly solving the differential equation. Furthermore, it also reduces the propagation of errors and prevents some stability issues. The non-parametric approximation using a linear combination of basis functions $\phi(x)$ is given by

$$\tilde{u}(x) = \sum_{i=1}^{K} \phi_i(x) c_i, \tag{2.4}$$

where c_i are the basis coefficients and K is the number of collocation points necessary to fit the data.

2.1.1 Methods to estimate parameters using function approximations

The simplest procedure to estimate parameters using basis expansion is known as the two-step method (VARAH, 1982; DUA, 2011). In this methodology, the coefficients of the base c_i are approximated considering only the fitting criteria

$$\min_{c_i} \quad \sum_{i=1}^{N} (\tilde{u}(x) - \hat{u}(x))^2, \tag{2.5}$$

and then in a separated stage, the parameters λ are estimated using the residue of the differential equation evaluated in N collocation points as

$$\min_{\lambda} \quad \sum_{i=1}^{N} (\mathscr{F}(x_1, ..., x_j, \tilde{u}, \frac{\partial \tilde{u}}{\partial x_1}, ..., \frac{\partial \tilde{u}}{\partial x_j}, ..., \frac{\partial^2 \tilde{u}}{\partial x_1 \partial x_j}, \lambda))^2, \tag{2.6}$$

where the evaluation of the residue in collocation points is equivalent to use a Monte Carlo integration method and only the values of λ are approximated maintaining fixed the coefficients of the basis expansion (MEER, 2019).

One difficulty of this procedure is that the function has to capture the behavior of the system without including the noise and uncertainties present in the observations. This issue can be solved by refining the number and position of the collocation points or including a penalty term in Eq. (2.5) to set a balance between overfitting and underfitting of the experimental data. The penalty term can be a high order derivative or, in methods such as principal differential analysis (POYTON et al., 2006), the residue of the differential equation as expressed in the following equation

$$\min_{c_i} \quad \sum_{i=1}^{N} (\tilde{u}(x) - \hat{u}(x))^2 + \alpha_r \sum_{i=1}^{N} (\mathscr{F}(x_1, ..., x_j, \tilde{u}, \frac{\partial \tilde{u}}{\partial x_1}, ..., \frac{\partial \tilde{u}}{\partial x_j}, ..., \frac{\partial^2 \tilde{u}}{\partial x_1 \partial x_j}, \lambda))^2, \tag{2.7}$$

where the weight α_r controls the amount of regularization and the residue of the model is computed using initial estimates of the parameters λ. In this case α_r can be also interpreted as an smoothing parameter that manages the fidelity of the approximation to the model (ZHANG; CAO; CARROLL, 2017). Poyton et al. (2006) developed a refined principal analysis, a method that employs Eq. (2.7) to estimate the coefficients of the basis functions and also the parameters of the model. This procedure is executed iteratively between the estimate of the coefficients and the parameters of the model until their estimates converge.

The work of Ramsay et al. (2007) provides important ideas to be considered in the development of other approximation strategies. They divide the variables estimated into two classes, one for the parameters of the model and the other for the coefficients of the basis function. This distinction is important because the optimization problem is directly concerned with the parameters of the model which the authors called as structural for their main importance. On the other hand, the coefficients of the function basis are designated as nuisance parameters due to their secondary role in the overall identification of the model. Another relevant difference between these two classes is that the number of nuisance parameters exceeds the structural parameters by a significant amount. The last argument is one of the reasons for the authors to avoid the estimation of both parameters at the same time. They use two different levels of optimization, in a similar way as explained before. In the inner level, the optimization only searches for better nuisance parameters and in the outer level, the structural parameters are refined using a different criterion. A third level in the optimization is possible to find the best

value of the weight in the penalized term but the authors adjusted it using some heuristics (CAO; RAMSAY, 2007).

Finally, we close our revision of the methods to estimate parameters in differential equations mentioning that the function basis expansion can be substituted for any function approximation procedure desired. In this work, we are interested in the application of neural networks as an approximator of the state variables of a model. Dua (2011) was one of the first works to introduce artificial neural networks using the two-step estimation approach and in Dua and Dua (2011), Raissi, Perdikaris and Karniadakis (2017b) were implemented simultaneous searches of structural and nuisance parameters in models described by ODEs and PDEs, respectively. In Chapter 4, we present an estimation methodology that combines some of the ideas presented in this subsection and employs the general structure proposed in Raissi, Perdikaris and Karniadakis (2017b).

2.2 Damage Model

Damage mechanics is a part of solid mechanics that allows a better understanding of the deterioration of materials and tries to predict its implication for mechanical integrity. Although damage involves the creation of microvoids and microcracks, discontinuities at a large scale of the medium, it has been introduced as a continuous variable that represents these volume and surface defects (LEMAITRE; DESMORAT, 2005).

Different strategies have been used in the development of models of damage, fracture and fatigue in elastic solids but few works have followed thermodynamically consistent frameworks. This study uses the damage and fatigue model developed by Boldrini et al. (2016), which addresses some mathematical deficiencies that have not been considered in previous works.

Boldrini et al. (2016) proposed a general thermodynamically consistent non-isothermal continuum framework for the evolution of damage, fatigue and fracture in materials under the hypothesis of small deformation. The approach followed is based on the use of conservation of mass, the principle of virtual power (PVP) and the first and second law of thermodynamics. In addition to the classical principles, it uses the phase field methodology to introduce fatigue and damage behavior. The kinematic descriptor for damage is a dynamic variable and its evolution is obtained from the PVP. The constitutive relations that define the governing equations of the model are expressed in terms of the free energy potential and the associated pseudopotential of dissipation for any given material (BOLDRINI et al., 2016; HAVEROTH et al., 2018).

The mentioned general framework is described below for a one-dimensional domain $\Omega = [a, b]$, linear elastic isotropic material, displacements only in the axial direction x, isothermal case and including only the effects of damage in the model. In such a situation, the model developed in Boldrini et al. (2016) consists of a coupled system of dynamic equations with the evolution of displacement u and damage phase field φ given by

$$\rho\frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\Big((1-\varphi)^2 E\frac{\partial u}{\partial x}\Big) + f_r(x), \tag{2.8}$$

$$\lambda_c\frac{\partial \varphi}{\partial t} = \frac{\partial}{\partial x}\Big(g_c\gamma_c\frac{\partial \varphi}{\partial x}\Big) + (1-\varphi)E\Big(\frac{\partial u}{\partial x}\Big)^2 - g_c\frac{\varphi}{\gamma_c}. \tag{2.9}$$

In these equations, g_c is the Griffith fracture energy, γ_c is a parameter associated to the width of the damage phase field layers, λ_c is related to the rate of damage change, E represents the Young's modulus and $f_r(x)$ is body load function. Eq. (2.8) describes the evolution of displacement u and Eq. (2.9) is the governing equation of the phase field damage. The variable φ represents the volumetric fraction of damaged material such that φ = 0 for virgin material, φ = 1 for fractured material and $0 < \varphi < 1$ for damaged material.

There are many possibilities for the boundary conditions of the governing equations. For the first equation, the displacement or the stress are given on the boundary. In the damage equation, the standard boundary condition is a homogeneous Neumann condition (null flux for φ at the boundary) (CHIARELLI et al., 2017).

2.3 Pseudo-experimental data : Numerical solution of the governing equations

In this work, we use pseudo-experimental data to validate the estimation procedure that will be proposed. The pseudo-experimental data provides the evolution of the state variables of the model, φ and u , from the solution of the governing equations with known parameters. The damage model is described by the system of governing differential equations Eq. (2.8) and Eq. (2.9) of hyperbolic and parabolic types, respectively. In both equations, the finite element method replaces the differential operator in space by a system of ordinary differential equations (RAGAB S. A., 2018). The assembled system is dependent on time and initial conditions in terms of u , $\dot{u}$ and φ at time $t = 0$ in the domain. At future times, the variables are approximated using the α-Method for the parabolic equation and the Newmark method for the hyperbolic equation.

The governing equations are coupled and therefore a strategy to relate their evolution progressively is necessary. An iterative semi-implicit scheme is implemented to accomplish this requirement. The damage φ at time step n is used to solve Eq. (2.8) for the displacement u at time step $n+1$; after that, the displacement u is used to calculate the strain $\dfrac{\partial u}{\partial x}$ and then Eq. (2.9) is solved. If the maximum damage in the model is less than 1, then the algorithm returns to the first step. The iterations start with the initial damage φ_0 (at time $t = 0$) as the first input to solve the displacement equation. Algorithm 1 resumes this procedure.

Algorithm 1 Iterative scheme to solve the governing equations

$\varphi \leftarrow \varphi_0$
$t \leftarrow 0$
while $max(\varphi) < 1$ **do**
  $u \leftarrow$ Solve Eq. (2.8) using the FEM and the Newmark method
  $\varphi \leftarrow$ Derive u and solve Eq. (2.9) using the FEM and the α- method
  $t \leftarrow t + \Delta t$
end while

In the next subsections, we explain how to solve the governing equations of the damage model following the steps and notation presented in Ragab S. A. (2018).

2.3.1 Solution of the damage equation

Consider a domain $a < x < b$ and the following initial and boundary conditions:

$$\varphi = \varphi_0, \qquad \text{at } t = t_0, \tag{2.10}$$

$$\frac{\partial \varphi}{\partial x} = 0, \qquad \text{at } x = a, \tag{2.11}$$

$$\frac{\partial \varphi}{\partial x} = 0, \qquad \text{at } x = b. \tag{2.12}$$

Multiplying Eq. (2.9) by a test function $v(x)$ and integrating over the domain, we obtain

$$\int_a^b v\Big(\lambda_c \frac{\partial \varphi}{\partial t}\Big) dx = \int_a^b v \left[\frac{\partial}{\partial x}\Big(g_c \gamma_c \frac{\partial \varphi}{\partial x}\Big) + (1-\varphi) E \Big(\frac{\partial u}{\partial x}\Big)^2 - g_c \frac{\varphi}{\gamma_c} \right] dx. \tag{2.13}$$

After integrating by parts the first term of the right side, Eq. (2.13) is rewritten as

$$\begin{aligned} \int_a^b v\Big(\lambda_c \frac{\partial \varphi}{\partial t}\Big) dx &= \left[v\Big(g_c \gamma_c \frac{\partial \varphi}{\partial x}\Big) \right]_{x=a}^{x=b} - \int_a^b \frac{\partial v}{\partial x}\Big(g_c \gamma_c \frac{\partial \varphi}{\partial x}\Big) dx \\ &+ \int_a^b v \left[(1-\varphi) E \Big(\frac{\partial u}{\partial x}\Big)^2 - g_c \frac{\varphi}{\gamma_c} \right] dx. \end{aligned} \tag{2.14}$$

Consider the following definitions:

$$R_a = g_c\gamma_c\Big(\frac{\partial\varphi}{\partial x}\Big)_{x=a}, \tag{2.15}$$

$$R_b = g_c\gamma_c\Big(\frac{\partial\varphi}{\partial x}\Big)_{x=b}. \tag{2.16}$$

Eq. (2.14) can be rewritten using Eqs. (2.15) and (2.16) as

$$\begin{aligned} 0 &= \int_a^b v\Big(\lambda_c\frac{\partial\varphi}{\partial t}\Big)dx - v(b)R_b + v(a)R_a + \int_a^b \frac{\partial v}{\partial x}g_c\gamma_c\frac{\partial\varphi}{\partial x}dx \\ &- \int_a^b v\left[(1-\varphi)E\Big(\frac{\partial u}{\partial x}\Big)^2 - g_c\frac{\varphi}{\gamma_c}\right]dx. \end{aligned} \tag{2.17}$$

Using the boundary conditions in Eqs. (2.11) and (2.12), the weak form of the damage equation is given by

$$\int_a^b v\Big(\lambda_c\frac{\partial\varphi}{\partial t}\Big)dx + \int_a^b \frac{\partial v}{\partial x}g_c\gamma_c\frac{\partial\varphi}{\partial x}dx - \int_a^b v\left[(1-\varphi)E\Big(\frac{\partial u}{\partial x}\Big)^2 - g_c\frac{\varphi}{\gamma_c}\right]dx = 0. \tag{2.18}$$

In an element level, the weak form is given by (2.17) without prescribed boundary conditions and with the integrals and derivatives evaluated at the coordinates of the element boundary x_0^e and x_n^e as

$$\begin{aligned} 0 &= \int_{x_0^e}^{x_n^e} v\Big(\lambda_c\frac{\partial\varphi}{\partial t}\Big)dx - v(x_n^e)R_{x_n^e} + v(x_0^e)R_{x_0^e} + \int_{x_0^e}^{x_n^e} \frac{\partial v}{\partial x}g_c\gamma_c\frac{\partial\varphi}{\partial x}dx \\ &- \int_{x_0^e}^{x_n^e} v\left[(1-\varphi)E\Big(\frac{\partial u}{\partial x}\Big)^2 - g_c\frac{\varphi}{\gamma_c}\right]dx. \end{aligned} \tag{2.19}$$

The approximation of $\varphi(x,t)$ in an element with m nodes and at any instant of time is given by

$$\varphi(x,t) \approx \bar{\varphi}^e(x,t) = \sum_{j=1}^{m}\bar{\varphi}_j^e(t)N_j^e(x) = \left[N^e(x)\right]\left\{\bar{\varphi}^e(t)\right\}, \tag{2.20}$$

where $\bar{\varphi}_j^e(t)$ represents the nodal values of φ at time t and $\left[N^e\right]$ is the matrix of shape functions. The dependence of the trial solution on x is given by $N_j^e(x)$ and the variation with time is given by $\bar{\varphi}_j^e(t)$.

Consequently, the time derivative and the gradient of the trial solution $\bar{\varphi}^e(x,t)$ are expressed, respectively, by

$$\frac{\partial \bar{\varphi}^e(x,t)}{\partial t} = \left[N^e(x)\right]\left\{\dot{\bar{\varphi}}^e(t)\right\}, \tag{2.21}$$

$$\frac{\partial \bar{\varphi}^e(x,t)}{\partial x} = \left[B^e\right]\left\{\bar{\varphi}^e(t)\right\}. \tag{2.22}$$

where $\left[B^e\right]$ is the matrix of the derivatives of the shape functions.

Now using the Galerkin method, the test function is defined as the linear combination of the basis functions $N_j(x)$. Following that, the approximation of $v(x)$ and its derivative in an element are written as

$$v(x) = \left\{d\right\}^T \left[N^e(x)\right]^T, \tag{2.23}$$

$$\frac{\partial v(x)}{\partial x} = \left\{d\right\}^T \left[B^e\right]^T. \tag{2.24}$$

Substituting Eqs. (2.20) to (2.24) into the weak form of the element in Eq. (2.19), and factoring out the terms $\left\{\bar{\varphi}^e\right\}$, $\left\{\dot{\bar{\varphi}}^e\right\}$ and $\left\{d\right\}^T$, we have

$$\begin{aligned}
&\left\{d\right\}^T \left[\int_{x_0^e}^{x_n^e} \lambda_c \left[N^e\right]^T \left[N^e\right] dx\right] \left\{\dot{\bar{\varphi}}^e\right\} + \left\{d\right\}^T \left[\int_{x_0^e}^{x_n^e} g_c \gamma_c \left[B^e\right]^T \left[B^e\right] dx\right] \left\{\bar{\varphi}^e\right\} \\
&+ \left\{d\right\}^T \left[\int_{x_0^e}^{x_n^e} \frac{g_c}{\gamma_c} \left[N^e\right]^T \left[N^e\right] dx + \int_{x_0^e}^{x_n^e} E \left[N^e\right]^T \left(\frac{\partial u}{\partial x}\right)^2 \left[N^e\right] dx\right] \left\{\bar{\varphi}^e\right\} \\
&+ \left\{d\right\}^T \left[N^e(x_0^e)^T R_{x_0^e} - N^e(x_n^e)^T R_{x_n^e} - \int_{x_0^e}^{x_n^e} E \left[N^e\right]^T \left(\frac{\partial u}{\partial x}\right)^2 dx\right] = 0.
\end{aligned} \tag{2.25}$$

The element matrices are defined as

$$\left[K^e\right] = \int_{x_0^e}^{x_n^e} g_c \gamma_c \left[B^e\right]^T \left[B^e\right] dx, \tag{2.26}$$

$$\left[M^e\right] = \lambda_c \int_{x_0^e}^{x_n^e} \left[N^e\right]^T \left[N^e\right] dx, \tag{2.27}$$

$$\left[M_u^e\right] = \int_{x_0^e}^{x_n^e} E \left[N^e\right]^T \left(\left[B^e\right] \left\{\bar{u}^e\right\}\right)^T \left(\left[B^e\right] \left\{\bar{u}^e\right\}\right) \left[N^e\right] dx, \tag{2.28}$$

$$\left[A^e\right] = \left[M_u^e\right] + \frac{g_c}{\gamma_c \lambda_c} \left[M^e\right] + \left[K^e\right], \tag{2.29}$$

and the element vectors are expressed by

$$\left\{F_u^e\right\} = \int_{x_0^e}^{x_n^e} E \left[N^e\right]^T \left(\left[B^e\right] \left\{\bar{u}^e\right\}\right)^T \left(\left[B^e\right] \left\{\bar{u}^e\right\}\right) dx, \tag{2.30}$$

$$\left\{R^e\right\} = \left[N^e(x_n^e)\right]^T R_{x_n^e} - \left[N^e(x_0^e)\right]^T R_{x_0^e}. \tag{2.31}$$

In terms of these matrices Eq. (2.25) can be rewritten as

$$\begin{aligned} 0 &= \left\{d\right\}^T \left(\left[M^e\right] \left\{\dot{\bar{\varphi}}^e\right\} + \left[K^e\right] \left\{\bar{\varphi}^e\right\} + \frac{g_c}{\gamma_c \lambda_c} \left[M^e\right] \left\{\bar{\varphi}^e\right\} \right. \\ &+ \left. \left[M_u^e\right] \left\{\bar{\varphi}^e\right\} - \left\{R^e\right\} - \left\{F_u^e\right\}\right). \end{aligned} \tag{2.32}$$

Since the test function $v(x)$ and consequently $\left\{d\right\}$ are arbitrary, the element equation using the auxiliary matrix from Eq. (2.29) can be finally written as

$$\left[M^e\right] \left\{\dot{\bar{\varphi}}^e\right\} + \left[A^e\right] \left\{\bar{\varphi}^e\right\} = \left\{R^e\right\} + \left\{F_u^e\right\}. \tag{2.33}$$

The assembled global system is an arrangement of coupled ordinary differential equations in time. The initial condition provided at $t = 0$ gives $\varphi(0)$ at all nodes, but for future times it is necessary to solve the assembled system given by the superposition of Eq. (2.33).

Considering the first order equation Eq. (2.33) and using the α-method, a weighted average of the derivatives at times steps $k+1$ and k is related with the slope of the chord as follows:

$$\frac{1}{\Delta t} \left[\left\{\bar{\varphi}^e\right\}^{k+1} - \left\{\bar{\varphi}^e\right\}^k\right] = \alpha \left\{\dot{\bar{\varphi}}^e\right\}^{k+1} + (1-\alpha) \left\{\dot{\bar{\varphi}}^e\right\}^k. \tag{2.34}$$

Using this relation for the first term in Eq. (2.33) , expressing $\left[M^e\right] \left\{\dot{\bar{\varphi}}^e\right\}$ at times k and $k+1$ and considering the mass and auxiliary matrices to be independent of time, we get

$$\left[\frac{1}{\Delta t}\left[M^e\right]+\alpha\left[A^e\right]\right]\left\{\bar{\varphi}^e\right\}^{k+1}=\left[\frac{1}{\Delta t}\left[M^e\right]-(1-\alpha)\left[A^e\right]\right]\left\{\bar{\varphi}^e\right\}^{k}+$$
$$\left[\alpha\left\{F_u^e\right\}^{k+1}+(1-\alpha)\left\{F_u^e\right\}^{k}\right]+\left[\alpha\left\{R^e\right\}^{k+1}+(1-\alpha)\left\{R^e\right\}^{k}\right]. \quad (2.35)$$

Introducing the following expressions:

$$\left[H^e\right]=\frac{1}{\Delta t}\left[M^e\right]+\alpha\left[A^e\right], \quad (2.36)$$

$$\left[T^e\right]=\frac{1}{\Delta t}\left[M^e\right]-(1-\alpha)\left[A^e\right], \quad (2.37)$$

$$\left\{Q^e\right\}=\alpha\left\{F_u^e\right\}^{k+1}+(1-\alpha)\left\{F_u^e\right\}^{k}+\alpha\left\{R^e\right\}^{k+1}+(1-\alpha)\left\{R^e\right\}^{k}, \quad (2.38)$$

the algebraic equation for an element is

$$\left[H^e\right]\left\{\bar{\varphi}^e\right\}^{k+1}=\left[T^e\right]\left\{\bar{\varphi}^e\right\}^{k}+\left\{Q^e\right\}. \quad (2.39)$$

The element equations are assembled into a global system enforcing continuity of damage at common nodes on the boundary of all elements and balance of derivatives given by Eqs. (2.15) and (2.16) in all the interior global nodes. Additionally, $\left\{F_u^e\right\}^{k}$ is considered only for the time k.

Finally the global system after application of boundary conditions is

$$\left[H\right]\left\{\bar{\varphi}\right\}^{k+1}=\left[T\right]\left\{\bar{\varphi}\right\}^{k}+\left\{F_u\right\}. \quad (2.40)$$

The α-method is unconditionally stable for $0.5 \leqslant \alpha \leqslant 1$ and take the values $\alpha = 0$ for Euler explicit, $\alpha = 1$ for Euler implicit and $\alpha = 0.5$ for Crank-Nicholson time marching schemes (RAGAB S. A., 2018).

2.3.2 Solution of the displacement equation

The approximated solution of Eq. (2.8) is obtained similarly to the damage equation. First, the finite element method is applied for the space operators and after that the coupled system of second-order ODEs is solved using a time marching scheme.

The approximation of $u(x,t)$ in an element with m nodes and at any instant of time is:

$$u(x,t) \approx \bar{u^e}(x,t) = \sum_{j=1}^{m} \bar{u}_j^e(t) N_j^e(x) = \left[N^e(x)\right] \left\{\bar{u}^e(t)\right\}, \tag{2.41}$$

where $\bar{u}_j^e(t)$ represents the nodal values of u at time t and $N_j^e(x)$ are the shape functions. The dependence of the trial solution on x is given by $N_j^e(x)$ and the variation with time is given by $\bar{u}_j^e(t)$ following the same arguments used in Eqs. (2.21) and (2.22).

Eqs. (2.28) and (2.30) already included the approximation of u in the term $\dfrac{\partial u}{\partial x}$ in a given element.

The following element equations are obtained after the determination of the weak form, use of Lagrange interpolation polynomials for the space approximation and application of the Galerkin method. Therefore,

$$\left[M_u^e\right] \left\{\ddot{\bar{u}}^e\right\} + \left[K_\varphi^e\right] \left\{\bar{u}^e\right\} = \left\{R_u^e\right\} + \left\{F^e\right\}. \tag{2.42}$$

The mass matrix of the inertia term is

$$\left[M^e\right] = \int_{x_0^e}^{x_n^e} \rho \left[N^e\right]^T \left[N^e\right] dx. \tag{2.43}$$

The stiffness matrix is given by

$$\left[K_\varphi^e\right] = \int_{x_0^e}^{x_n^e} E\Big(1 - \left[N^e\right] \left\{\bar{\varphi}^e\right\}\Big)^2 \left[B^e\right]^T \left[B^e\right] dx. \tag{2.44}$$

The forcing vector is

$$\left\{F^e\right\} = \int_{x_0^e}^{x_n^e} f_r(x) \left[N^e\right]^T dx \tag{2.45}$$

and the vector R_u is written as

$$\left\{R_u^e\right\} = \left[N^e(x_n^e)\right]^T R_u^{x_n^e} - \left[N^e(x_0^e)\right]^T R_u^{x_0^e}, \tag{2.46}$$

where the terms $R_u^{x_j^e}$ are

$$R_u^{x_0^e} = \left[E(1-\varphi)^2 \Big(\frac{\partial u}{\partial x} \Big) \right]_{x=x_0^e}, \tag{2.47}$$

$$R_u^{x_n^e} = \left[E(1-\varphi)^2 \Big(\frac{\partial u}{\partial x} \Big) \right]_{x=x_n^e}. \tag{2.48}$$

The assembling of the global system of equations is developed following the same considerations described in the solution of the damage equation. The system of ODEs is given by

$$\Big[M_u\Big] \Big\{\ddot{\bar{u}}\Big\} + \Big[K_\varphi\Big] \Big\{\bar{u}\Big\} = \Big\{R_u\Big\} + \Big\{F\Big\}. \tag{2.49}$$

In order to solve the system of ODEs in time, the Newmark time marching scheme is applied. Considering the Taylor series expansion of the function $\bar{u}(t)$ about time level k, we have the approximation

$$\bar{u}(k+1) \approx \bar{u}(k) + \dot{\bar{u}}(k)\Delta t + \frac{1}{2}\ddot{\bar{u}}(k+\gamma)(\Delta t)^2, \tag{2.50}$$

where

$$\ddot{\bar{u}}(k+\gamma) = (1-\gamma)\ddot{\bar{u}}(k) + \gamma\ddot{\bar{u}}(k+1). \tag{2.51}$$

Combining Eqs. (2.50) and (2.51) and using them in the displacement vector, we obtain

$$\Big\{\bar{u}\Big\}^{k+1} = \Big\{\bar{u}\Big\}^{k} + \Big\{\dot{\bar{u}}\Big\}^{k} \Delta t + \frac{(1-\gamma)}{2} \Big\{\ddot{\bar{u}}\Big\}^{k} (\Delta t)^2 + \frac{\gamma}{2} \Big\{\ddot{\bar{u}}\Big\}^{k+1} (\Delta t)^2. \tag{2.52}$$

Following a similar approach for the velocity approximation, we have

$$\Big\{\dot{\bar{u}}\Big\}^{k+1} = \Big\{\dot{\bar{u}}\Big\}^{k} + (1-\alpha)\Delta t \Big\{\ddot{\bar{u}}\Big\}^{k} + \alpha\Delta t \Big\{\ddot{\bar{u}}\Big\}^{k+1}, \tag{2.53}$$

where α and γ are parameters not equal necessarily.

Writing Eq. (2.49) at time step $k+1$ and solving for the acceleration, we find

$$\Big\{\ddot{\bar{u}}\Big\}^{k+1} = \Big(\Big[M_u\Big]\Big)^{-1} \left[\Big\{R_u\Big\} + \Big\{F\Big\} - \Big[K_\varphi\Big] \Big\{\bar{u}\Big\}^{k+1} \right]. \tag{2.54}$$

Substituting Eq. (2.54) into Eq. (2.52),

$$\left\{\bar{u}\right\}^{k+1} = \left\{\bar{u}\right\}^{k} + \left\{\dot{\bar{u}}\right\}^{k} \Delta t + \frac{(1-\gamma)}{2}\left\{\ddot{\bar{u}}\right\}^{k} (\Delta t)^2 + \frac{\gamma}{2}(\Delta t)^2 \Big(\left[M_u\right]\Big)^{-1} \left[\left\{R_u\right\} + \left\{F\right\} - \left[K_\varphi\right]\left\{\bar{u}\right\}^{k+1}\right]. \quad (2.55)$$

Finally rearranging terms, we arrived at

$$\Big(\left[M_u\right] + \frac{\gamma}{2}(\Delta t)^2 \left[K_\varphi\right]\Big)\left\{\bar{u}\right\}^{k+1} = \left[M_u\right]\left\{\bar{u}\right\}^{k} + \left[M_u\right]\left\{\dot{\bar{u}}\right\}^{k} \Delta t + \frac{(1-\gamma)}{2}\left[M_u\right]\left\{\ddot{\bar{u}}\right\}^{k} (\Delta t)^2 + \frac{\gamma}{2}(\Delta t)^2 \Big(\left\{R_u\right\} + \left\{F\right\}\Big). \quad (2.56)$$

With displacement and acceleration at time k, the solution at time step $k+1$ can be determined as follows:

1. Solve Eq. (2.56) for $\left\{\bar{u}\right\}^{k+1}$;
2. Solve the system Eq. (2.54) equation for $\left\{\ddot{\bar{u}}\right\}^{k+1}$;
3. Compute $\left\{\dot{\bar{u}}\right\}^{k+1}$ using Eq. (2.53).

The parameters α and γ have an important role in the stability and accuracy of the Newmark time marching scheme. For second-order accuracy, the values recommended are $\alpha = 0.5$ and $\gamma = 0.5$ (RAGAB S. A., 2018).

3 Artificial Neural Networks

Artificial neural networks are a popular subset of machine learning methods in realizations of artificial intelligence, i.e., broader range of tasks that requires human cognition, such as object detection, pattern recognition, and image analysis. Some of the reasons that explain the rapid growth of this technique, in comparison with other machine learning algorithms, are its simplicity, continuous improvement when the volume of data increases and the new hardware resources available for its application.

Neural networks are mathematical constructs that are inspired by the brain capacity of humans and animals to perform complex tasks without much effort. However, this machine learning system only has some similarities with the actual functioning of the brain that resembles the behavior and structure that humans have developed. A network is organized in layers made up of interconnected processing units where each connection is weighted and receives a transformation by an activation function. The learning process is developed through experiences that are presented as training examples and the strength of the connections consolidates the knowledge acquired by the neural network. Although this method is conceptually simple, it is possible to approximate non-linear relationships and complex patterns found in diverse applications (MEER, 2019; CHAKRAVERTY; MALL, 2017).

This work solves an identification problem based on the application of a feedforward neural network as a function approximator. The feedforward architecture has been explored in the solution of differential equations due to the possibility to compute analytic expressions of its derivatives using backward propagation. Recent advances in computational techniques, such as automatic differentiation, have expanded this potential because now it is possible to define networks with multiple hidden layers, i.e., deep learning, and automatically compute their derivatives using the record of their operations. As will be presented in this chapter, the simplicity of the feedforward architecture and the use of automatic differentiation allow an intuitive implementation of physics informed neural networks for solution and identification of differential equations. This paradigm reduces the amount of training data required to solve forward and inverse problems and it produces solutions that follow physical laws trough a mathematical model.

This chapter presents some fundamental concepts about neural networks, such as the general model of a neuron and the basic neural network architectures. In addition, conventional definitions from machine learning systems are used to contextualize the learning paradigms and tasks that a neural network can perform. Most of the content involves the description of important components of feedforward neural networks to finally introduce the new paradigm that enforces physics laws during the learning process.

3.1 General model of a neuron and network architectures

An artificial neuron is the basic processing unit that transforms a set of inputs into a single output. In a general neuron model, the inputs k connected to the neuron j are multiplied by a weight ω_{jk}, then this linear map is shifted using a constant b_j known as bias and finally, an activation function σ applies a transformation on the whole term (see Eq. (3.1)). Several models of neurons, such as perceptron and sigmoid, use the same basic elements to compute their output. The differences between these models are in the activation function employed and the interpretation given to the bias. The structure used to connect the neurons is defined through the architecture of the network. Although many architectures are applied in research and practice, there are three general categories for neural networks architectures: multilayer feedforward, convolutional and recurrent. These structures are described in the next subsections.

3.1.1 Multilayer feedforward neural networks

In this architecture the layers are typically fully connected, which means that every neuron in a layer is connected to every other neuron in the contiguous layer. Connections between the same or previous layers in the block are not allowed which means that there is no feedback communication during the forward computation of an output. The simplest feedforward structure is a single-layer network where the input layer directly feed their signals in the output.

Multilayer feedforward networks add blocks of neurons between the input and output layers that are typically known as hidden layers. These layers help to detect relationships and patterns during the training. However, it is necessary to balance their number against the training time of the network. Fig. 3.1 illustrates a feedforward neural network referred to as 2-3-4-1 because it has 2 inputs, 3 neurons in the first hidden layer, 4 neurons in the second hidden layer and finally 1 output.

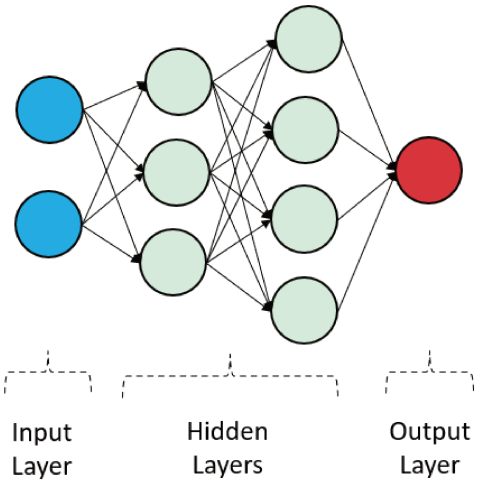

Figure 3.1 – Architecture of a feedforward artificial neural network.

3.1.2 Convolutional neural networks

A neural network can be composed of layers assuming or ignoring a special structure in its inputs. A convolutional layer assumes that there exists an important relation between close inputs, something common for data given through images. Although there are some differences between the general model of a neuron previously presented and the computation applied in convolutional layers, the main change is that the assumption of spatial relations in the inputs allows a reduction in the number of parameters (HAYKIN; HAYKIN, 2009; RAMSUNDAR; ZADEH, 2018).

Convolutional networks are known for their successful performance in visual tasks, e.g. image, and video applications, but they can also be applied in other tasks, such as natural language processing or time series models. In Fig. 3.2 we show a typical convolutional network architecture generated using the NN-SVG tool (LENAIL, 2019). The network has an input of 28x28 pixels and its architecture is composed of convolutional, max-pooling, and dense layers. The legends in the top give the number of feature maps followed by their size, and in the bottom, the main operation of each layer is presented. For example, after the first convolution there are 15 featured maps each one of 24x24 pixels (15@24x24).

A convolution operation is an element-wise multiplication and addition between a local region of the input and a filter. The purpose of a convolutional layer is to extract useful features and this is achieved by applying different filters moved along the input. The output of a convolutional layer has a number of channels (feature maps) equal to the number of filters and its size is

usually smaller than the size of the input.

A pooling layer also applies a filter over its input, but with the objective of discovering invariant features through dimensionality reduction. Pooling layers return the average (Average-Pool) or the maximum (Max-Pool) value over each local region traversed by its filter. This layer forces the learning of relevant patterns independently of their positions in the input. In max-pooling layers, filters do not have learnable parameters and they only change the size without affecting the number of channels.

Fully connected layers, also known as dense layers, are usually employed in the final part of the convolutional networks. A dense layer takes its input from a higher-level layer and discovers the relations to perform an accurate classification in the network output.

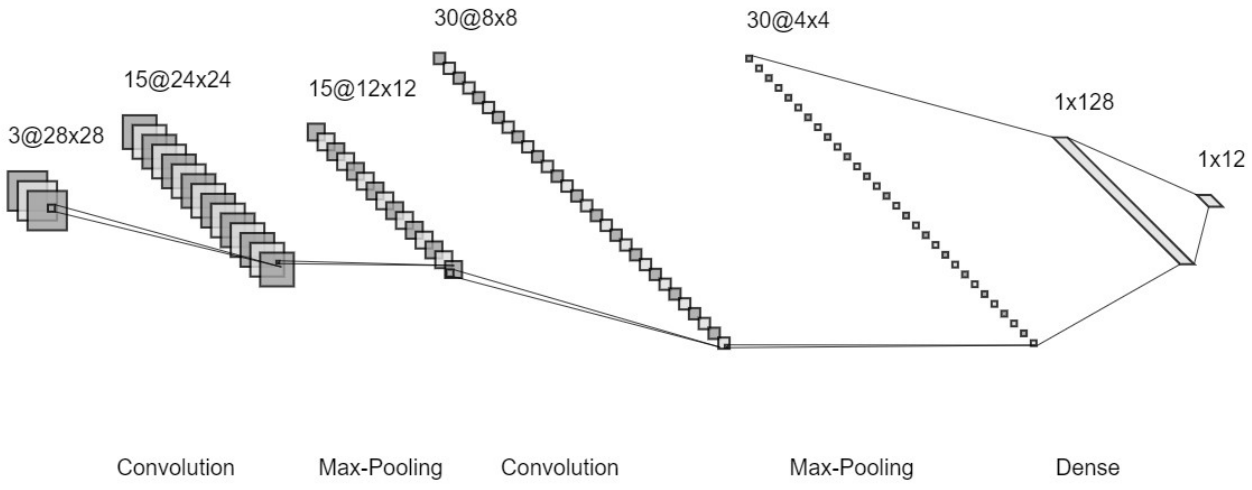

Figure 3.2 – Convolutional neural network.

3.1.3 Recurrent neural networks

This architecture differs from the feedforward networks in that connections among neurons permit feedback communication in the network as presented in Fig. 3.3. The forward propagation in recurrent networks takes into account information perceived previously in time, developing a certain type of memory. The activation of a hidden state at a given time depends on a weighted sum of the inputs in the current state and additionally includes a weighted sum of the hidden state of previous steps. Recurrent networks are suitable to recognize patterns in sequences of data, such as numerical time series data, music generation, sentiment classification, machine translation, and other important applications.

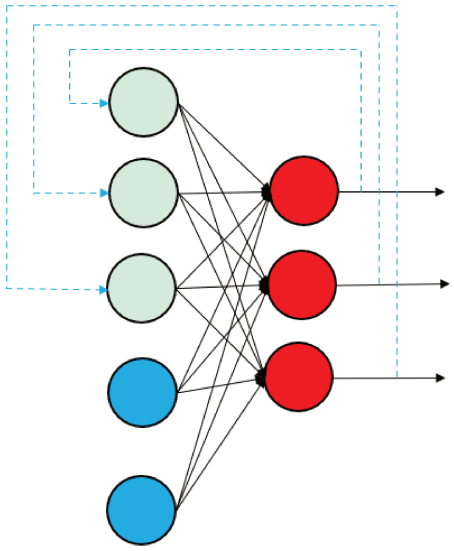

Figure 3.3 – Architecture of a recurrent neural network.

3.2 Learning paradigms and tasks

Machine learning algorithms are based on information that usually comes from observations of the environment where they operate. This experience provides a set of examples, also called training data or samples, and its attributes guide the selection of an appropriate learning process. The three learning paradigms, namely supervised, unsupervised and reinforcement, are characterized by the way that the network learns.

In supervised learning, the available training set includes the desired outputs, also known as labels. The network adapts its behavior influenced by the difference between labels and outputs. Additionally, an algorithm optimizes the neural network model using mathematical techniques that lead this difference towards a minimum value.

Reinforcement learning does not have a single correct answer. Instead, the learning process is based on an agent or critic that learns from its experiences. These experiences are formed by positive (reward) and negative (penalty) stimuli from the environment. Its aim is to learn sequences of actions that will lead an agent to maximize an objective function (HAYKIN; HAYKIN, 2009) .

In unsupervised learning, no labels are given to the learning algorithm and the process does not receive stimulus from their environment. The purpose of this learning process can be grouping similar elements from the unlabeled data or determine hidden patterns in the inputs of the

network.

In recent years, neural networks have been successfully applied in tasks of association, clustering, pattern recognition and predictions in areas such as science, engineering, agriculture, finance, energy, marketing, and other important fields. The diverse areas of applications show the potential and universality of neural networks to solve different kinds of problems (ABIODUN et al., 2018). The learning tasks are commonly classified as pattern association, pattern recognition, and function approximation, where the last two groups have the largest number of applications in engineering and science. In pattern recognition tasks, a neural network takes input information and produce a discrete output that assigns it to a category or class. Function approximation tasks are useful in applications that require a nonlinear mapping between an input and a continuous output (HAYKIN; HAYKIN, 2009). The ability to produce nonlinear continuous mappings is exploited in system identification problems, inverse modeling and solution of differential equations as will be explained at the end of this chapter.

3.3 Multilayer feedforward networks for function approximation tasks

In this work, we are interested in a supervised learning process for function approximation of one or more continuous outputs that describe the behavior of a physical variable. The basic structure of the neural networks selected for our application consists of an input layer that communicates with a block of one or more hidden layers using a system of weighted connections and biases. At the end of the network, the last hidden layer links to an output layer that provides an approximated value of the expected response. There are two simple operations applied to each neuron. The first operation is a weighted sum for all the incoming values and an addition of a bias. Following that, an activation function σ applies a nonlinear transformation which gives the actual output of the neuron. The strength of connections among neurons is defined by parameters θ (weights and biases) that are learned after a training process using labeled data.

In this work, we use a feedforward fully connected architecture as presented in Fig. 3.4. Neurons in a given layer l receive input from all the neurons in the previous layer $l-1$ and feed their output to all the neurons in the next layer $l+1$.

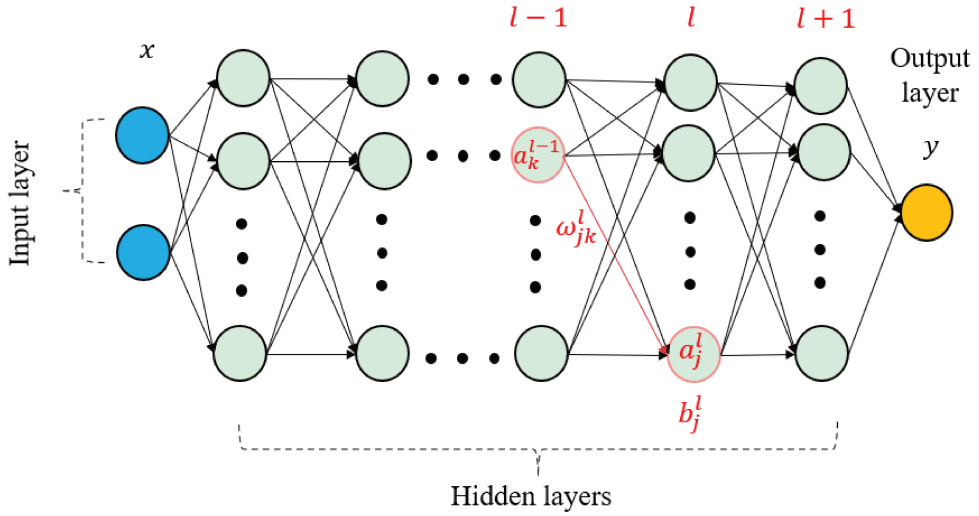

Figure 3.4 – Multilayer feedforward architecture used in this work.

The computation of the neural network output is known as forward propagation. The activation a_j^l of the j^{th} neuron of a layer l is related to the activations in the layer $l-1$ by

$$a_j^l = \sigma_l(\sum_k^{n^{l-1}} \omega_{jk}^l a_k^{l-1} + b_j^l), \tag{3.1}$$

where the sum is over all neurons k of the layer $l-1$ and the terms in this equation follow the notation used in Nielsen (2015):

- ω_{jk}^l is the weight for the connection from the k_{th} neuron in the $(l-1)_{th}$ layer to the j_{th} neuron of the l_{th} layer;
- b_j^l denotes the bias of the j_{th} neuron of the l_{th} layer;
- a_j^l represents the activation of the j_{th} neuron in the l_{th} layer;
- a_k^{l-1} represents the activation of the k_{th} neuron in the $(l-1)_{th}$ layer.

A different activation function can be used in each layer and there are several options to apply this non-linear transformation.

There is another important computation where the error signal passes leftward through the network. This left or backward pass is commonly known as backward propagation. It is a recursive process where the weights and bias of the neural network change in accordance with the sensitivity of the output to these parameters.

3.4 Activation functions

Each neuron can have a different activation function, but it is common to use the same function for all the neurons in a given layer. Non-linearity is the main characteristic desired in an activation function because it allows the representation of complex trends in data and is the basis of the universal approximation capabilities of neural networks. The only requirement that an activation function has to satisfy is differentiability, but other aspects as the range and behavior of the derivatives are also important for the stability and learning speed of the method (GULIKERS, 2018; HAYKIN; HAYKIN, 2009). In the following subsections, some of the common activation functions used to solve function approximation tasks are described.

3.4.1 Sigmoid

The sigmoid function is given by

$$\sigma(z) = \frac{e^z}{1 + e^z} \tag{3.2}$$

provides an activation with a range between 0 and 1, has continuous derivatives and gives a smooth relation between the input and output of a neuron. However, the values assigned to the weights can drive their response to a saturated region where the derivative is very small. Consequently, if the activation is in this region, the learning will be slow (NIELSEN, 2015).

3.4.2 Hyperbolic tangent

This function is considered as a rescaled and symmetric version of the sigmoid activation and given by

$$\sigma(z) = \frac{e^{2z} - 1}{e^{2z} + 1}. \tag{3.3}$$

In this case, the amplitude of output lies inside the range -1 and 1 and is centered about 0. Despite having larger derivatives than the sigmoid function, the hyperbolic tangent function may also lead to saturation problems.

3.4.3 Identity

This function has an unrestricted range and a constant derivative equal to one,

$$\sigma(z) = z. \tag{3.4}$$

Its input is not modified in the transformation and as a consequence can be interpreted as a linear regression operator (GULIKERS, 2018).

3.4.4 ReLU

The rectified linear unit works as an identity function for positive values and it is deactivated for values less than zero. Therefore,

$$\sigma(z) = \max(0, z). \tag{3.5}$$

As a result, its range is given by all positive numbers and their derivative is equal to one. Although is less intuitive to see, this function can also approximate nonlinear behaviours but does not have a well defined derivative when its input is zero. ReLU is a function widely used in many applications because it does not have saturation problems but some complications can arise when many activations are zero.

In Fig. 3.5 the behavior described for some non-linear activation functions and their first derivatives are shown.

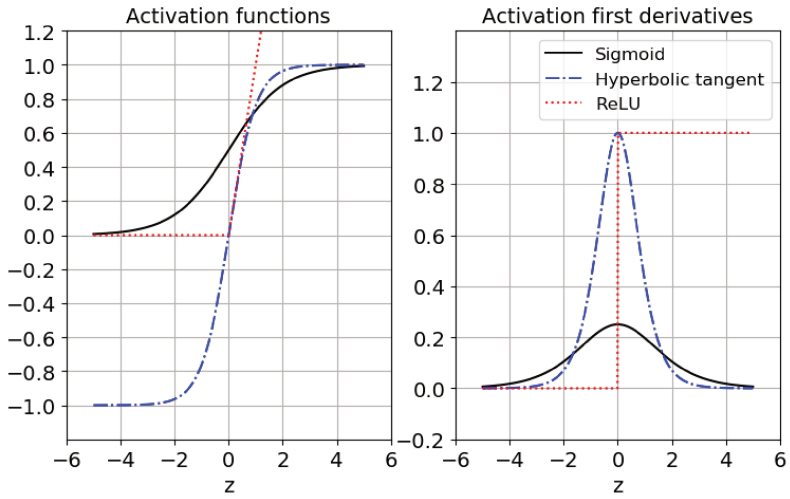

Figure 3.5 – Non-linear activation functions and their first derivatives.

3.5 Loss functions

In Section 3.2, we described how in supervised learning, inputs and label outputs are given as training data and the learning process is basically concerned with the adjustment of differences between desired outputs $\hat{y}$ and values computed for the neural network y. The loss function can be interpreted as a measure of how good is a prediction model and its formulation defines how these differences or errors are calculated.

The loss of a neural network model is the objective function in the optimization process and, in order to use backpropagation, two conditions are recommended for its selection: First, it is necessary to write the function as an average of the training samples. Second, it should be possible to establish a relationship between the output y and the parameters ω and b of the network (NIELSEN, 2015).

Different functions can be implemented depending on the type of the problem addressed. In the next subsections, some common options used in regression tasks are briefly described.

3.5.1 Mean squared

The mean square function is the most common selection in regression tasks, and given by

$$L = f(\hat{y}_i, y_i) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i)^2, \tag{3.6}$$

where N is the number of samples or points considered. This function sums the square distance of the predicted and target value and returns an average of this accumulated error.

3.5.2 Mean absolute

This loss function is the sum of the absolute difference of target values and outputs computed in the network, i.e,

$$L = f(\hat{y}_i, y_i) = \frac{1}{N}\sum_{i=1}^{N}|\hat{y}_i - y_i|. \tag{3.7}$$

Although the mean absolute is considered more robust to outliers in data sets, constant values of the gradient near the minima create difficulties during the learning process.

3.5.3 Huber loss

Huber loss can be considered as a smoothed version of the mean absolute function that becomes a mean square when the difference of the target and the predicted values are small. Therefore,

$$L = \frac{1}{N}\sum_{i=1}^{N} L_i \tag{3.8}$$

where

$$L_i = \begin{cases} \frac{1}{2}(\hat{y}_i - y_i)^2, & \text{if } |\hat{y}_i - y_i| \leqslant \delta \\ \delta|\hat{y}_i - y_i| - \frac{\delta^2}{2}, & otherwise. \end{cases}, \tag{3.9}$$

As the mean absolute, the Huber loss function has a better sensitivity to outliers than mean square; its disadvantage is the inclusion of the smoothing hyperparameter δ that needs to be tuned.

3.5.4 Log hyperbolic cosine

This function can be considered as an improved version of the Huber loss function, and is written as

$$L = \frac{1}{N}\sum_{i=1}^{N} \log(\cosh(\hat{y}_i - y_i)). \tag{3.10}$$

It computes the loss using the logarithm of the hyperbolic cosine of the prediction error. The log hyperbolic cosine function has the advantages of the Huber loss function but, additionally, it is twice differentiable which makes possible to use optimization methods that need second derivatives.

3.6 Backward propagation and automatic differentiation

The learning process of a neural network takes place when an algorithm tries to minimize an objective function that measures the difference between target and computed outputs. In this optimization process, the computed output is adjusted until the loss function reaches a defined tolerance or the algorithm stops after a maximum number of iterations.

As presented in Section 3.3, the predicted output is an implicit result of the propagation of previous activations and is dependent on the weights and bias that define the network. Therefore, to solve the optimization problem is important to find the effect of these parameters on the computed output. From calculus, we know that this information is given by the gradient of the loss function and backpropagation is an easy way to go back from the output to the input and obtain these derivatives using the chain rule (NIELSEN, 2015).

Consider a single-layer network with only one neuron j in the output. We can rewrite the loss function in terms of the difference between the label and predicted output as follows:

$$L = f(e_j), \tag{3.11}$$

$$e_j = \hat{y}_j - y_j, \tag{3.12}$$

$$y_j = a_j^o = \sigma(z_j^o), \tag{3.13}$$

$$z_j^o = \sum_k \omega_{jk}^o a_k^{o-1} + b_j^o = \sum_k \omega_{jk}^o a_k^i + b_j^o, \tag{3.14}$$

where f is one of the loss functions described in Section 3.5, k represents the number of inputs in the single-layer network, the superscript o symbolizes the output layer and in this case the layer $(o-1)$ is the input layer i.

According to the rules for differentiating compositions of functions, the derivative of the loss function L in terms of the neural network parameters is expressed as

$$\frac{\partial L}{\partial \omega_{jk}^o} = \frac{\partial L}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial z_j^o}\frac{\partial z_j^o}{\partial \omega_{jk}^o} = \delta_j^o a_k^i, \tag{3.15}$$

$$\frac{\partial L}{\partial b_j^o} = \frac{\partial L}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial z_j^o}\frac{\partial z_j^o}{\partial b_j^o} = \delta_j^o, \tag{3.16}$$

where the term δ_j^o, known as the local gradient, is

$$\delta_j^o = \frac{\partial L}{\partial e_j}\frac{\partial e_j}{\partial y_j}\frac{\partial y_j}{\partial z_j^o} = \frac{\partial L}{\partial y_j}\sigma'(z_j^o). \tag{3.17}$$

Similarly, in the case where j is a hidden node in a multilayer network with one hidden layer and only one neuron in the output , we have the following forward propagation:

$$z_j^l = \sum_k \omega_{jk}^l a_k^i + b_j^l, \tag{3.18}$$

$$a_j^l = \sigma(z_j^l), \tag{3.19}$$

$$z_m^o = \sum_j \omega_{mj}^o a_j^l + b_m^o, \tag{3.20}$$

$$y_m = a_m^o = \sigma(z_m^o), \tag{3.21}$$

$$e_m = \hat{y}_m - y_m, \tag{3.22}$$

$$L = f(e_m), \tag{3.23}$$

where k are the number of inputs in the layer i, j is a neuron in the hidden layer l and m is the neuron in the output.

The derivative of the loss function L with respect to ω_{jk}^l is computed using the chain rule as

$$\frac{\partial L}{\partial \omega_{jk}^l} = \frac{\partial L}{\partial e_m}\frac{\partial e_m}{\partial y_m}\frac{\partial y_m}{\partial z_m^o}\frac{\partial z_m^o}{\partial a_j^l}\frac{\partial a_j^l}{\partial z_j^l}\frac{\partial z_j^l}{\partial \omega_{jk}^l}. \tag{3.24}$$

The previous expression is rewritten in a compact form

$$\frac{\partial L}{\partial \omega_{jk}^l} = \delta_m^o \omega_{mj}^o \sigma'(z_j^l) a_k^i = \delta_j^l a_k^i, \tag{3.25}$$

using the following relations:

$$\delta_m^o = \frac{\partial L}{\partial e_m}\frac{\partial e_m}{\partial y_m}\frac{\partial y_m}{\partial z_m^o}, \tag{3.26}$$

$$\frac{\partial z_m^o}{\partial a_j^l} = \omega_{mj}^o, \tag{3.27}$$

$$\frac{\partial a_j^l}{\partial z_j^l} = \sigma'(z_j^l), \tag{3.28}$$

$$\frac{\partial z_j^l}{\partial \omega_{jk}^l} = a_k^i, \tag{3.29}$$

$$\delta_j^l = \delta_m^o \omega_{mj}^o \sigma'(z_j^l). \tag{3.30}$$

It is possible to define the gradient of the loss function following the implicit relations of weights, bias, and previous activations. The same steps can be easily extended for any configuration due to the simple mathematical structure of neural networks. Backpropagation is based on a recursive approach where each layer is considered separately and then is linked with the derivatives inside of a neuron and the activations of previous layers (HAYKIN; HAYKIN, 2009).

Automatic differentiation is a family of similar techniques to backpropagation used in physics and engineering fields and in recent years have been also applied to machine learning implementations. This methodology follows the standard computations of a given program and calculates their derivatives using an overall composition of the basic derivatives and the chain rule. There are two principal accumulation modes and different implementations available which are covered and explained in more depth in Baydin et al. (2015).

3.7 Optimization algorithms in neural networks

The optimization algorithm searches for better learnable parameters θ using an iterative procedure where small changes improve the predicted outputs. Some of the difficulties in this process arise when a loss function is non-convex and problems such as local minima and saddle points are encountered during the training time.

Most of the algorithms used in practical applications are based on the gradient descent method and have been inspired to cope with the problems previously described. Apart from the first-order optimization algorithms, there is another category based on the use of second-order derivatives. These methods are faster when the derivatives are easily calculated but also have trouble dealing with local minima and saddle points. In the following subsections, three important optimization algorithms extensively used in machine learning are summarized.

3.7.1 Gradient descent method

Considering that the loss function L is continuously differentiable, making possible to compute its gradient ∇L, the unconstrained optimization problem is defined as

$$\min_{\theta \in R^n} L(\theta). \tag{3.31}$$

We are interested in a stationary point of L, that is, an argument θ^* at which the condition of optimality $\nabla L(\theta^*) = 0$ is satisfied.

In gradient descent methods, the direction vector $v_i = -\nabla L(\theta_i)$ is selected as the search direction, which is the direction of steepest descent.

A variable η is introduced as the step-size (known as the learning rate) and the parameters at iteration $i + 1$ can be expressed by

$$\theta_{i+1} = \theta_i - \eta \nabla L(\theta_i). \tag{3.32}$$

If we define the correction

$$\Delta\theta = \theta_{i+1} - \theta_i, \tag{3.33}$$

we have,

$$\Delta\theta = -\eta \nabla L(\theta_i). \tag{3.34}$$

The loss function at $L(\theta_{i+1})$ is approximated using a first-order Taylor series, as

$$L(\theta_{i+1}) = L(\theta_i) + \nabla L^T \Delta\theta. \tag{3.35}$$

Substituting Eq. (3.34) in Eq. (3.35), we finally arrive at

$$L(\theta_{i+1}) = L(\theta_i) - \eta||\nabla L||^2. \tag{3.36}$$

Eq. (3.36) shows that for a positive learning rate η, the loss function is decreasing (HAYKIN; HAYKIN, 2009) .

As mentioned previously, some of the challenges of optimization procedures come from the existence of local minima solutions and saddle points. For first-order methods, a proper selection of the learning rate is the key in the optimization process, but it is not enough to avoid these complications. As a result, alternative algorithms have included ideas as the division of data and training using subsets, the inclusion of previous gradient information and adaptive learning rates.

The following subsection presents a gradient-descent based method that merges some of the best ideas considered for alternative gradient descent optimization algorithms.

3.7.2 Adam method

The adaptive moment estimation method combines different gradient descent algorithms (Momentum and RMSprop) using the history of gradients to move faster in the relevant direction and also to adapt the learning rate of the optimization process.

This method computes the weighted averages of gradients m_i and squared gradients q_i as

$$m_i = \beta_1 m_{i-1} + (1 - \beta_1) g_i, \tag{3.37}$$

$$q_i = \beta_2 q_{i-1} + (1 - \beta_2) g_i^2, \tag{3.38}$$

where g_i represents the gradients of the loss function $L(\theta_i)$ and the hyperparameters β_1 and β_2 control the exponential decay rates of these moving averages.

A correction for the bias during the initial iterations is added as

$$\hat{m_i} = \frac{m_i}{1 - \beta_1}, \tag{3.39}$$

$$\hat{q_i} = \frac{q_i}{1 - \beta_2}. \tag{3.40}$$

Finally , the Adam optimization rule is given by

$$\theta_{i+1} = \theta_i - \frac{\eta}{\sqrt{\hat{q}_i} + \epsilon}\hat{m}_i, \tag{3.41}$$

where a fixed term ϵ is included to avoid division by zero and it is necessary to tune two additional hyperparameters β_1 and β_2.

3.7.3 L-BFGS

In Newton methods, second-order derivatives of the objective function give a better understanding of the function topology and allow the selection of a more efficient path for the search procedure.

Consider a second-order Taylor series expansion of the loss function,

$$L(\theta_{i+1}) = L(\theta_i) + \nabla L^T \Delta\theta + \frac{1}{2}\Delta\theta^T \nabla^2 L \Delta\theta. \tag{3.42}$$

The change in the loss function is defined by

$$\Delta L = L(\theta_{i+1}) - L(\theta_i) = \nabla L^T \Delta\theta + \frac{1}{2}\Delta\theta^T \nabla^2 L \Delta\theta. \tag{3.43}$$

To minimize ΔL, we differentiate the last expression with respect to $\Delta\theta$, i.e,

$$\nabla L + \nabla^2 L \Delta\theta. \tag{3.44}$$

From this expression, we update θ in the next step $i + 1$ using

$$\Delta\theta = -(\nabla^2 L)^{-1} \nabla L, \tag{3.45}$$

and

$$\theta_{i+1} = \theta_i - (\nabla^2 L)^{-1} \nabla L. \tag{3.46}$$

Newton's method has a long execution time and requires large storage space to compute the Hessian matrix $\nabla^2 L$. In contrast, quasi Newton methods circumvent this problem by directly

approximating the inverse of the Hessian matrix using the first order derivatives. Although these methods have good convergence rates and in many applications outperform gradient descent algorithms, in large-scale optimization problems, such as neural networks, quasi Newton methods also need significant storage space (ROBITAILLE et al., 1996).

There are different strategies for the estimation of the Hessian matrix in quasi-Newton methods and one of the most popular is the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS). The BFGS Hessian approximation can be based on the full history of gradients or only on more recent gradients. The last case is known as limited memory BFGS, abbreviated as L-BFGS, which significantly reduces the storage requirement.

At the ith iteration, the BFGS method computes a new iterate by the formula

$$\theta_{i+1} = \theta_i - \alpha_i H_i \nabla L(\theta_i), \tag{3.47}$$

where α_i is the step size, ∇L is the gradient of the loss function and H_i is the inverse approximation of the Hessian matrix.

This matrix is approximated at each step using the following equation:

$$H_{i+1} = (I - c_i s_i y_i^T) H_i (I - c_i y_i s_i^T) + c_i s_i s_i^T, \tag{3.48}$$

where

$$s_i = \theta_{i+1} - \theta_i, \tag{3.49}$$

$$y_i = \nabla L(\theta_{i+1}) - \nabla L(\theta_i), \tag{3.50}$$

$$c_i = \frac{1}{y_i^T s_i}. \tag{3.51}$$

In the L-BFGS method instead of storing the entire matrix H_i, the set of m most recent curvature pairs (s_k, y_k) are saved and used to update H_i. As a result, it is not necessary to construct and store the dense inverse Hessian approximation at each time step (BERAHAS; JAHANI; TAKáč, 2019; NOCEDAL; WRIGHT, 2006).

3.8 Physics informed neural network

A physics informed neural network (PINN) is a methodology that employs physics laws to reduce the amount of data needed in the learning process of a neural network model (RAISSI; PERDIKARIS; KARNIADAKIS, 2017a; RAISSI; PERDIKARIS; KARNIADAKIS, 2017b). This method can be used to solve partial differential equations (PDEs) but is also very useful in the solution of inverse problems.

In the general case, the governing equations of a dynamic system are given in the following form:

$$\frac{\partial u}{\partial t} = \mathscr{L}(u, \lambda), \tag{3.52}$$

subject to the Dirichlet and Neumann boundary conditions

$$u(x,t) = g(x,t), \qquad x \in \partial\Omega_D, \tag{3.53}$$

$$K\frac{\partial u}{\partial x} = q(x,t), \qquad x \in \partial\Omega_N, \tag{3.54}$$

and with initial condition

$$t = 0: \qquad u(x,0) = u_0(x), \qquad x \in \Omega. \tag{3.55}$$

Here $u(x,t)$ denotes the solution of the PDE, $\mathscr{L}$ is a function or differential operator parametrized by λ, Ω is the domain and K comes from a given constitutive relation.

It is also important to define the residue of the PDE, which plays an important role in the training process, by

$$R := \frac{\partial u}{\partial t} - \mathscr{L}(u, \lambda). \tag{3.56}$$

3.8.1 Types of physics informed network models

Raissi, Perdikaris and Karniadakis (2017a) proposed two types of approximations, a continuous PINN where space and time are inputs of the neural network, and a discrete time model, where space is the only input and the time derivative is expressed using a classical time integration method.

3.8.1.1 Continuous PINN

In this model, time and space (t, x) are inputs of the PINN and the key elements common in any Artificial Neural Network (ANN) model are also used in the construction of the continuous PINN:

- Training data: in forward problems (solution of PDEs), the data used to train the network come from boundary and initial conditions, Eqs. (3.53) to (3.55), and is complemented with the physics laws given by the PDE. These principles are incorporated through evaluation of the residue, Eq. (3.56), in a set of collocation points randomly selected in the domain. In an inverse problem (discovery of PDEs), data of the solution is available in a space-time domain and used with Eq. (3.56) to approximate the solution $u(x, t)$ and estimate the parameters λ of the operator $\mathscr{L}$ (RAISSI; PERDIKARIS; KARNIADAKIS, 2017a).
- Activation function: hyperbolic tangent or sigmoid functions are commonly used as activation in all hidden layers and an identity activation in the output layer. The hyperbolic tangent easily captures nonlinear trends and has a symmetric range and the identity function is strategically used in the last layer to restore the range and scale of the approximated function $u(x, t)$.
- Loss function: any function written as an average of the individual cost of the training samples can be used. The loss function of PINNs not only depends on the output value of the network, but also on the residue, boundary and initial conditions in the case of forward problems.
- Training algorithm: First-order or second-order methods can be used to optimize the loss expression of the model as described in Section 3.7.
- Hyperparameters: Some settings considered in the algorithm developed for the PINNs are the ANN architecture, the initialiser of weights and biases and the normalization of the inputs. There is no rule to tune these hyperparameters but their selection can certainly affect the performance of the PINN.

PINNs approach using continuous models for data driven solution and discovery of PDEs can be summarized as follows:

1. Define hyperparameters: initializer, input normalization, ANN architecture, number of samples for training, optimization methods, and other relevant settings.
2. Generate the training data:

- forward solution: use boundary and initial conditions.
- inverse solution: take samples from the available solution in the spatio-temporal domain.

3. Select collocation points to evaluate the residue.
4. Forward propagation.
5. Loss function:
   - forward solution: using the output of the network $u(x, t)$, evaluate initial and Dirichlet conditions and calculate the residue and the Neumann boundary conditions.
   - inverse solution: construct the loss function using computed outputs, sample solutions and the evaluation of the residue.
6. Learning with automatic differentiation or backpropagation: optimize the PINN model using the sensitivity of the loss in terms of learnable parameters.

3.8.1.2 Discrete PINN

In this PINN is possible to introduce a numerical method to approximate the partial time derivative of Eq. (3.52). Consequently, in a discrete PINN, there is only a spatial dependence in the construction of a neural network.

The Runge-Kutta (RK) implicit scheme was the numeric method adopted by Raissi, Perdikaris and Karniadakis (2017a) and the number of RK stages becomes an important hyper-parameter for the construction of the multioutput neural network. Although the key elements explained for the case of continuous PINNs are similar, there are some minor differences due to the introduction of the RK method, the use of multioutput networks (to approximate the stages of RK) and other considerations taken in the construction of a discrete time model.

3.8.2 Solution of differential equations using a continuous PINN

As presented in the literature review, the work of Lagaris, Likas and Fotiadis (1998) was one of the first papers to propose the solution of differential equations using neural networks. Following their approach, the solution of a PDE is constructed considering a trial solution that satisfies the initial and boundary conditions.

Consider the following trial solution:

$$u^{tr} = G(x,t) + F(x,t,N(x,t,\theta)), \tag{3.57}$$

where the term $N(x,t,\theta)$ represents the output of the neural network model and the terms $G(x,t)$ and $F(x,t,N)$ are defined to satisfy the boundary and initial conditions by construction.

In this approach the loss function L of the neural network is computed in a set of random collocation points (x_i,t_i) inside the domain , and is defined in terms of the PDE residue as

$$R_i(u^{tr}) = \frac{\partial u^{tr}(x_i,t_i,\theta)}{\partial t} - \mathscr{L}(u^{tr},\lambda). \tag{3.58}$$

$$L = f(\hat{R}_i, R_i) = \frac{1}{N_c}\sum_{i=1}^{N_c} f(\hat{R}_i, R_i), \tag{3.59}$$

where f represents one of the functions described in Section 3.5 and N_c is the number of collocation points. In order to approximate the solution of the differential equation, the label values $\hat{R}_i$ are zero in all the N_c collocation points.

Nowadays, the terms in Eq. (3.58) are automatically computed in machine learning frameworks, but without the help of these tools, it was necessary to derive the expressions to evaluate the residue.

The physics informed methodology takes the trial solution directly from the output of the neural network model,

$$u^{tr} = N(x,t,\theta), \tag{3.60}$$

and constructs the loss L of the model considering terms for the residue L_r, boundary conditions L_b and initial conditions L_i, i.e,

$$L = L_r + L_b + L_i, \tag{3.61}$$

where

$$L_r = \frac{1}{N_c}\sum_{i=1}^{N_c} f(\hat{R}_i, R_i), \tag{3.62}$$

$$L_b = \frac{1}{N_b}\sum_{i=1}^{N_b} f(\hat{u_i}, u_i^{tr}) + \frac{1}{N_b}\sum_{i=1}^{N_b} f\Big(\frac{\partial \hat{u_i}}{\partial x}, \frac{\partial u_i^{tr}}{\partial x}\Big), \tag{3.63}$$

$$L_i = \frac{1}{N_i} \sum_{i=1}^{N_i} f(\hat{u}_i, u_i^{tr}), \tag{3.64}$$

N_b and N_i are the boundary and the initial conditions points, respectively. The label values in equations Eqs. (3.63) and (3.64) are computed using the Eqs. (3.53) to (3.55).

3.8.3 Discovery of differential equations using a continuous PINN

The PINN methodology is also employed in the discovery of the parameters λ in the differential operator $\mathscr{L}(u, \lambda)$ of Eq. (3.52). Using some observations of the variable $\hat{u}_i$ (distributed in the spatial-temporal domain) and the partial differential equation (from a physics model), it is possible to estimate the parameters λ that reproduces the physical behavior given by the data.

The physics informed model uses the same trial solution given in Eq. (3.60) but now the residue of the PDE is a function of the output of the neural network and also of the unknown parameters λ as

$$R_i(u^{tr}, \lambda) = \frac{\partial u^{tr}(x_i, t_i, \theta)}{\partial t} - \mathscr{L}(u^{tr}, \lambda). \tag{3.65}$$

The parameters λ that define the PDE equation of the model are learnable parameters similar to θ, but they are external to the neural network. The loss function of the physics informed model is given by two contributions,

$$L = L_r + L_c, \tag{3.66}$$

where L_r is the residue loss and L_c is the collocation loss. In identification problems, the solution of the governing equations is partially known and the collocation loss can be interpreted as the criterion to fit the output of the neural network model to the pseudo-experimental data. Its training points N_c are usually adopted as the same collocation points used for the residue loss and this loss is evaluated as follows:

$$L_c = \frac{1}{N_c} \sum_{i=1}^{N_c} f(\hat{u}_i, u_i^{tr}). \tag{3.67}$$

An important aspect of this formulation is that the collocation loss only affects the learning process of the parameters θ. Conversely, the residue loss has influence in the adjustment of parameters θ and λ.

4 Identification approach and results

The previous chapters presented some fundamental concepts to identify the material parameters of the damage model. We described some estimation methods in differential equations, gave an overview of the damage model, outlined the methods to obtain the pseudo-experimental data and finally summarized the artificial neural networks and the physics informed methodology. This chapter collects all this information for a gradual identification of the damage model parameters and presents the results obtained with this approach. It also includes an explanation of how a physics informed neural network is used to solve this specific problem, some considerations taken and the discussion of results.

4.1 Physics informed neural network for identification of parameters

The governing equations presented in Section 2.2 give the evolution of displacement and damage for a linear elastic material. In order to explore the use of neural networks in this model identification problem, different physical cases are considered following the assumptions taken for the final equations of the model. The identification is primarily based on the damage evolution given that all the parameters from the phase field methodology appear in this expression. This consideration also reduces the complexity of the neural network model to only one parabolic partial differential equation.

We define our data-driven methodology using a feedforward neural network that takes space and time values as inputs and returns a continuous function of damage in the space-time domain. This output is employed to compute the residues of the partial differential equation and the boundary and initial conditions.

As we presented in chapter 2, there are different approaches to estimate parameters in differential equations. In most of these approaches, authors have considered the fitting of the experimental data as the main objective of the problem and penalized the residue of the governing equations. We treat the identification of the parameters as a multi-objective optimization problem where it is necessary to achieve a good balance among each of the terms that appear in the loss function of our neural network model. In the physics informed methodology, the terms in the loss function are simultaneously minimized without consideration of its role in the identification. This can increase the difficulty of material parameter estimation since the number of learning parameters in the neural network exceed by far the material parameters of the physics model. On

the other hand, our implementation use weights α_j for each term L_j in the total loss expression L stated as,

$$L = \alpha_r L_r + \alpha_i L_i + \alpha_b L_b + \alpha_c L_c, \tag{4.1}$$

where L_r, L_i, L_b and L_c are the residue, initial, boundary and collocation losses, respectively. It is natural to assume that the residue loss is more relevant in the estimation of the material parameters by its mathematical definition. While the residue loss is written from the governing equations of the model, the collocation loss is defined using the output of the neural network. This means that the corrections of the neural network parameters are directly affected by L_c and, on the other hand, the material parameters depend heavily on L_r.

We use a few training simulations to tune the weights α_j considering their current contribution to the total loss. In most cases, this process is straightforward and produces good results. Other works based on the PINN classical model have already proposed the introduction of parameters to represent this relative importance of the loss function terms and observed significant improvements in their results (MEER, 2019). Another difference with the classical PINN methodology is that we include the boundary and initial conditions of the model because in some cases it facilitates the search of the neural network parameters.

4.1.1 Hyperparameters and optimization strategy

In neural network models, learnable parameters θ are progressively adjusted using the backpropagation algorithm. Hyperparameters, in contrast, are chosen using expert intuition. A typical search of hyperparameters includes the number of neurons and layers, loss and activation functions, amount of training data, methods for weight initialization, batch size and learning rate of gradient descent, number of iterations kept in memory for the L-BFGS optimization, among others. The selection of hyperparameters defines the set of configurations that maximize a metric associated with the accuracy of a neural network model. We know from previous works that the order of the residue loss function is a metric closely related with the capacity of the network to approximate a PDE solution. However, in identification problems a low value of this metric does not guarantee a good estimation of the material constants λ_j.

After some tests using different configurations, we tune the hyperparameters using literature recommendations and practical experiences. We use the mean absolute function for the collocation loss L_c and the mean square function for the residue L_r, boundary conditions L_b and initial condition L_i losses. We employ a tangent hyperbolic function as the default activation and the Glorot normal method to initialize the learnable weights of the model (GLOROT;

BENGIO, 2010). In contrast to common activation functions, the hyperbolic tangent is preferred because its non-linearity is complemented with a symmetrical distribution and greater derivatives. Furthermore, the Glorot initialization helps to avoid vanishing and exploding gradient problems.

For the optimization algorithms, we set the following hyperparameters:

- Adam: $\alpha = 1 \times 10^{-3}, \beta_1 = 0.9, \beta_2 = 0.999$.
- L-BFGS: 50 corrections for the limited memory and the convergence criterion to stop the iterations is $(L^k - L^{k+1})/max(|L^k|, |L^{k+1}|, 1) \leq ftol$, where k is a given iteration and $ftol = 1 \times 10^{-12}$.

During the tuning process, it was observed that the inclusion of the residue demands the use of networks with many layers and neurons to avoid poor estimates of the material constants. Possible reasons for this situation are local minima solutions, initial values defined for the search, bounds given in the optimization algorithms and hyperparameters used during the minimization. Although the penalizing constants regularize the search (RAMSAY et al., 2007), we combine some of the ideas of the two-step method (DUA, 2011), the principal differential analysis (POYTON et al., 2006) and the generalized smoothing approach (RAMSAY et al., 2007) to address these difficulties. Our implementation employs the following three stages in the training:

- **First stage**: we use an initial L-BFGS algorithm with a maximum of 1000 iterations. In this stage, only the collocation loss L_c is optimized and the aim is to refine the initial values of learnable parameters using the pseudo-experimental data available.
- **Second stage**: the second stage is a batch gradient descent optimization of L using the Adam algorithm with a number of steps between 5000 and 20000. In this stage, we explore a combination of Adam and L-BFGS methods to reduce the occurrence of local minima solution. The strategy consists of training the network using the Adam algorithm, and after a defined number of iterations, performing the L-BFGS optimization of the collocation loss L_c with a small limit of executions.
- **Third stage**: the training is complemented with a stage of simultaneous optimization using the L-BFGS method but this time with a maximum of 20000 iterations and a small tolerance for the convergence criterion.

We observed that in some cases this strategy allowed the use of networks with a smaller number of layers and neurons, and consequently, this approach is applied in all cases.

Some conjectures initially considered were that a higher amount of training data with refinement around the concentration of damage would provide better approximation results. However, it was noticed that an amount of approximately 10% of available data was enough to obtain good results without any improvement with larger percentages. Similarly, the results were not affected when the sampling had more points near the initial damage. Fig. 4.1 shows a random distribution of data in a solution domain.

Using the proposed optimization strategy in combination with the default configurations selected for most of the hyperparameters, it is possible to reduce the number of elements to be tuned. Finally, the number of neurons, layers, and the batch size of the Adam optimization are selected in a random search. This not only narrows the search but also lessens the necessity of expert intuition.

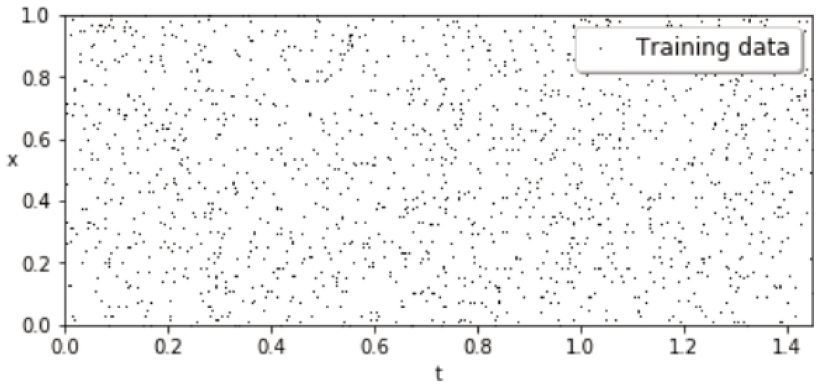

Figure 4.1 – Random distributed data in collocation loss.

4.1.2 General methodology

Though we use the core of the PINN methodology, we implemented some changes considering ideas from works that apply neural networks to solve differential equations and also from estimation strategies that employ other function approximators. A short description of the principal elements of our implementation is summarized as follows:

1. A deep feedforward neural network is created to compute the damage evolution φ. The network takes space x and time t as inputs.

2. The output of this network is derived using automatic differentiation which allows the computation of the PDE residue R. The strain is part of the pseudo-experimental data generated using the forward solution of the partial differential equations following the steps described in Section 2.3.

3. We use a loss expression L composed of 4 important contributions: collocation points, residue, boundary and initial conditions.

4. The contributions in the loss function are weighted using their relative importance α_j. We interpret some terms as constraints of a multi-objective optimization process.

5. The training process starts with a short optimization of the collocation loss L_c and after this, a combination of Adam and L-BFGS is used to minimize the total loss expression L and L_c , respectively. Finally, we define an additional optimization stage where the neural network parameters θ are adjusted, and the material parameters of the governing equation λ_j are identified simultaneously.

The methodology applied is represented in Fig. 4.2 with each part of the diagram related to the previous description. In spite of the fact that only the damage is approximated using the neural network, there is a coupling between damage and displacement equations that needs to be addressed. In the following sections, we present how this relation was expressed and some results obtained.

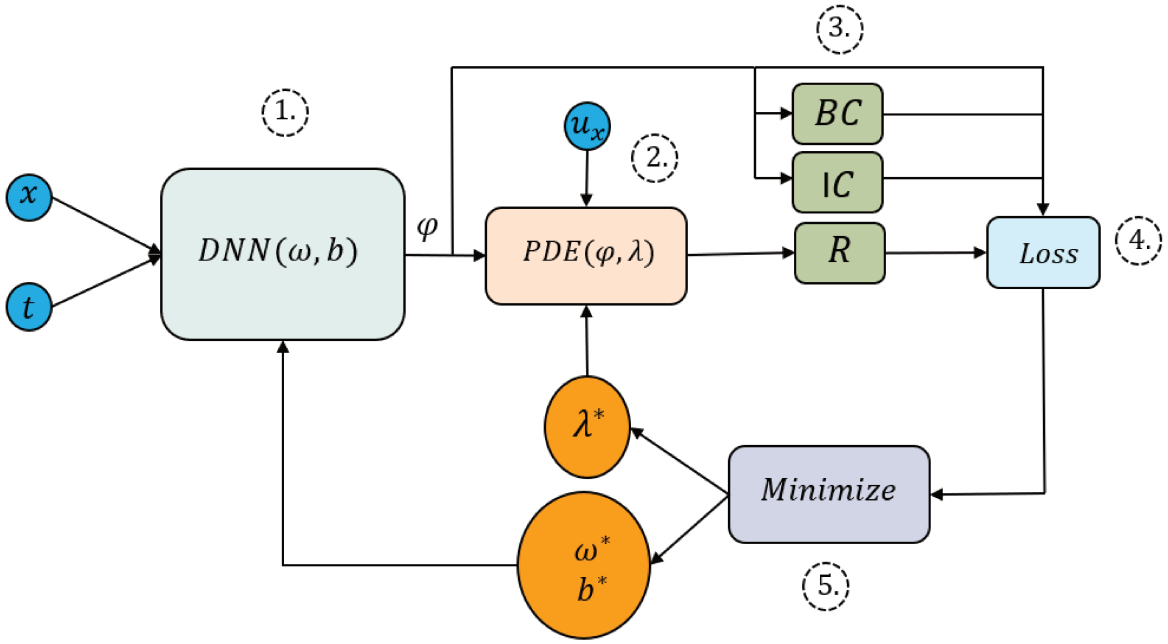

Figure 4.2 – PINN for parameter identification of the damage equation.

4.2 Identification considering a constant strain in the bar

The first implementation of PINNs for parameter identification is developed considering a decoupling of damage and displacement equations. In order to do that, it is assumed a constant

strain in the spatial domain to compute the damage independently of the displacement evolution. This assumption reduces the mathematical complexity of the problem and gives the possibility to compare the solution with another identification approach.

The first case proposed is given by a bar fixed on the left side $(x = 0)$ with zero initial displacement and velocity in the spatial domain which is shown in Fig. 4.3. A bar of length $L = 1\,\mathrm{m}$, cross sectional area $A = 1 \times 10^{-3}\,\mathrm{m}^2$ and Young's modulus $E = 72\,\mathrm{GPa}$ subjected to a constant strain of $1\ \%$ is considered.

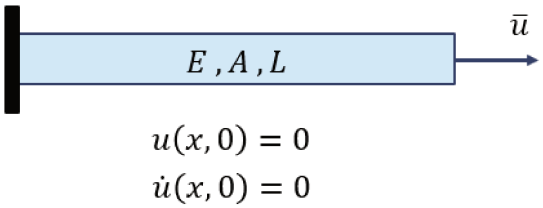

Figure 4.3 – Bar under constant normal strain.

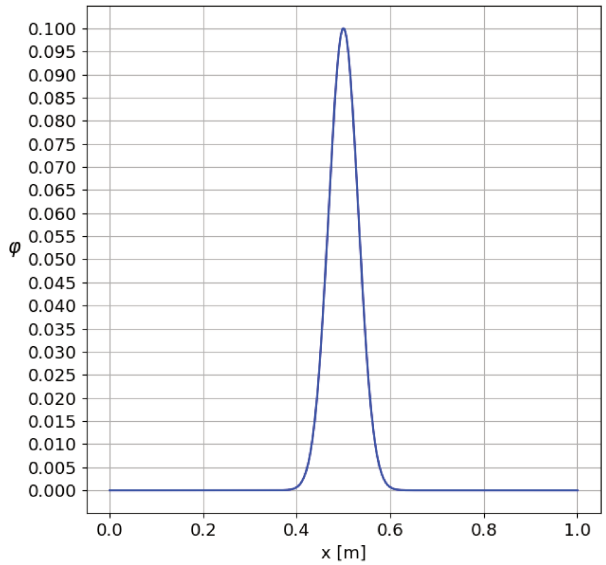

Figure 4.4 – Initial damage for the bar with the constant strain.

In the case of the damage equation, the boundary conditions were presented in Eqs. (2.11) and (2.12) (homogeneous Neuman conditions), and the initial condition presented in Fig. 4.4 is a concentrated damage $\varphi_0 = 0.1$ in the middle of the bar $\left(x = 0.5L\right)$ introduced using a Gaussian function.

The damage evolution in terms of parameters λ_1 (g_c), λ_2 (λ_c) and λ_3 (γ_c) can be rewritten

as

$$\frac{\partial \varphi}{\partial t} = \left(\frac{\lambda_1 \lambda_3}{\lambda_2}\right) \frac{\partial}{\partial x}\left(\frac{\partial \varphi}{\partial x}\right) + \left(\frac{E^*}{\lambda_2}\right)(1-\varphi) - \left(\frac{\lambda_1}{\lambda_2 \lambda_3}\right)\varphi, \tag{4.2}$$

where E^* represents the product of Young's modulus E and the squared strain u_x^2.

The pseudo-experimental data for this identification was obtained from a forward solution using the methods and parameters given in Table 4.1.

Table 4.1 – Classical numerical solution for the bar with the constant strain.

Space discretization	Interpolation order	Nodes	Time marching	Δt	t_f
FEM	1	1000	Crank-Nicolson	$2 \times 10^{-4}\,\mathrm{s}$	$0.303\,\mathrm{s}$

The neural network was implemented using the open-source framework TensorFlow (ABADI et al., 2015) with the Python application software interface (API). The methodology implemented for material identification is based on a simple mathematical and computational structure that takes advantage of the tools and comprehensive libraries available in TensorFlow.

Table 4.2 presents the configurations that provided the best results. The architecture of the neural network is given by 2 inputs, 3 hidden layers with 13 neurons in each layer and an output layer with 1 neuron. Each stage of the optimization process has a defined number of maximum iterations and a total of 5000 (x, t) pairs are used to compute the residue and collocation losses. The penalizing weights in the total loss function L are $\alpha_r = 20$, $\alpha_i = 10$, $\alpha_b = 2$, $\alpha_c = 1$. The first optimization stage employs the L-BFGS method, with a limit execution of 1000 iterations, to minimize the collocation loss. Then, in the second stage, a loop of 10000 iterations trains the total loss with the Adam method. Within this loop, each 500 iterations the collocation loss is minimized with an L-BFGS method with 80 executions at most. In the last stage the total loss is trained again with the L-BFGS method. The iteration limits of the first and second optimization stages are chosen empirically. These limits are lower considering that when the method alternates between the total and collocation losses there will be repetitive variations with a general decreasing trend. Thus, the role of these initial stages is to provide better estimates for the final optimization stage due to their trade-off in the search of the nuisance and structural parameters.

Table 4.2 – Hyperparameters of the PINN for the bar under constant strain.

Layers and neurons	Samples	L-BFGS	Batch Size	Adam + L-BFGS	L-BFGS
[2,13,13,13,1]	5000	1000	2000	10000-80(500)	20000

The percentage errors for the identification of the material parameters are presented in Table 4.3 and the neural network approximation of damage at different times using this material

constants is presented in Fig. 4.5. The training time is about 8 minutes and the implementation ran in a GTX 1050 GPU.

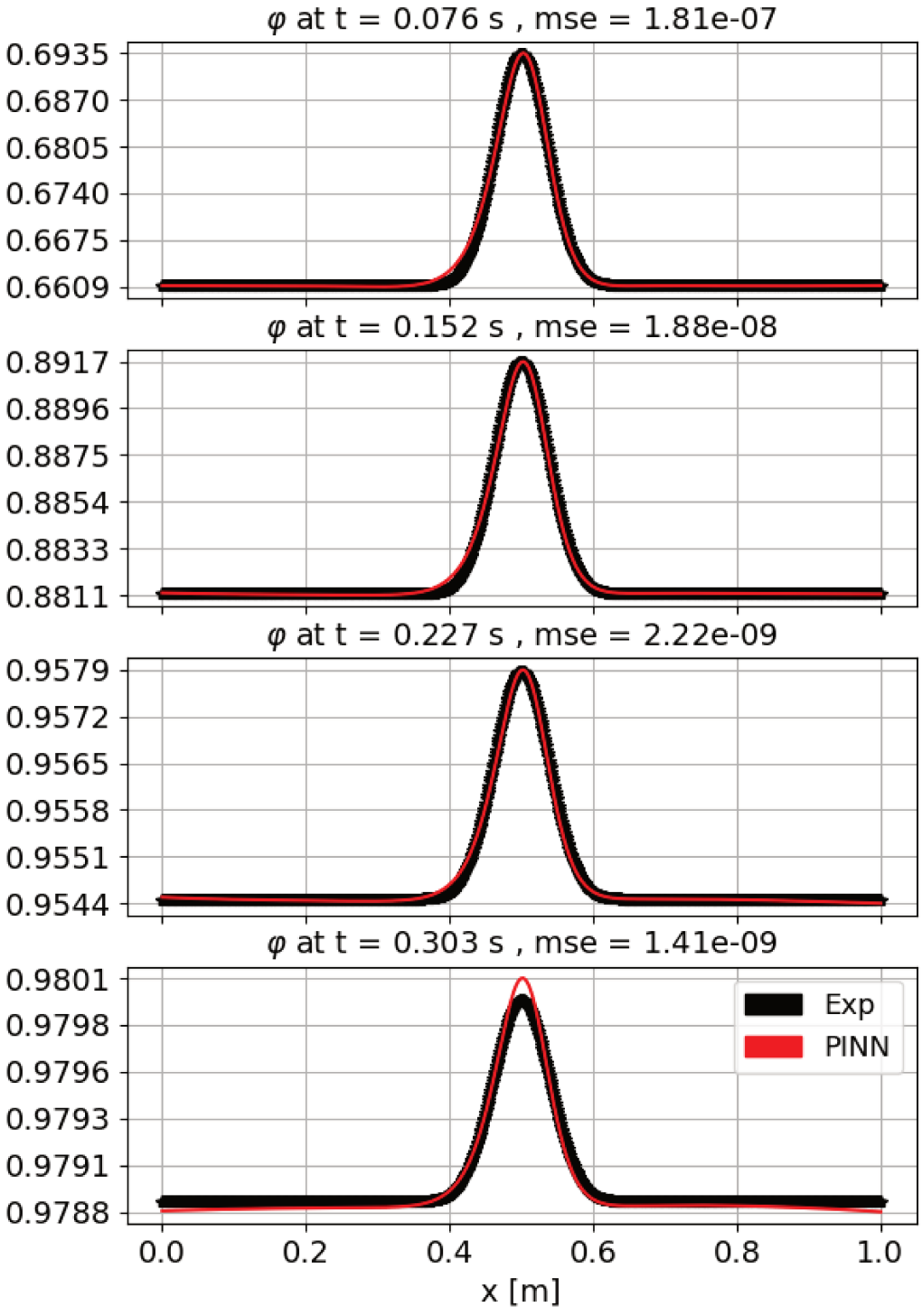

Figure 4.5 – Damage approximation for a bar under a constant strain.

Table 4.3 – Percentage errors in the identification using the PINN for the bar under constant strain.

Parameter	Label value	Estimated value	Percentage error
λ_1	3.90×10^3	3.98×10^3	2.08%
λ_2	5.00×10^5	5.01×10^5	0.22%
λ_3	6.00×10^{-2}	6.11×10^{-2}	1.87%

The results considering only the evolution of damage showed small percentage errors for all the parameters in the damage equation. Similarly, it can be seen in Fig. 4.5 that the neural network approximation was accurate in comparison with the pseudo-experimental data provided. There were only slight differences in the last step ($t = 0.303$) of damage evolution but, in general, the neural network reproduced the qualitative and quantitative behavior given by the damage equation of the model. Using this methodology, we considered a smaller number of hyperparameters in the tuning process, and with a total of 417 learnable parameters, it was possible to approximate the damage behavior and obtain a good estimation of the material parameters.

Figs. 4.6 to 4.8 present the learning curves for each optimization stage. As described in Section 4.1, the metric for the collocation loss is the mean absolute error and the mean square function is used for the other terms.

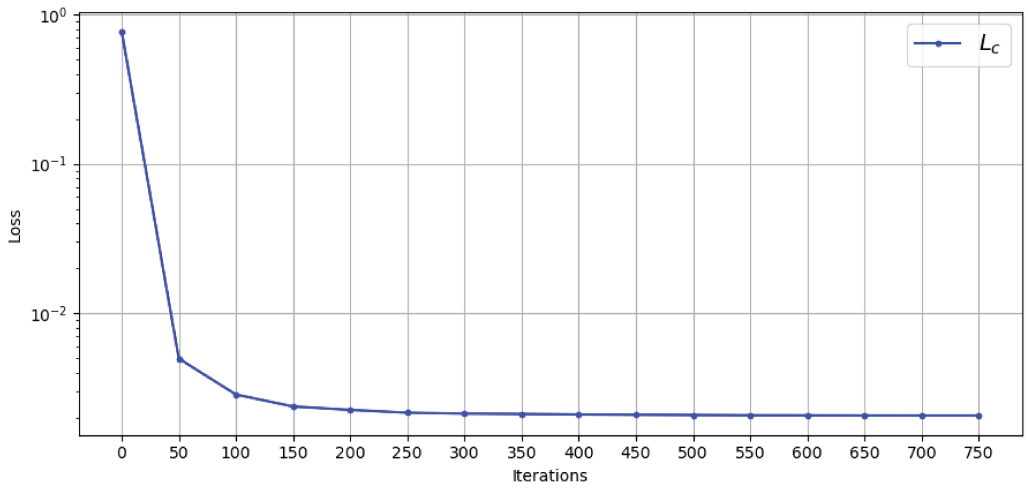

Figure 4.6 – Learning curve for the first optimization stage.

Fig. 4.6 shows how the collocation loss plunges in the initial 50 iterations of the first stage and then remains steady. The seasonality observed in Fig. 4.7 is caused by the execution of the L-BFGS method (every 500 iterations) to minimize only the collocation loss. It can be

noticed that the peaks in the total loss L coincide with the troughs of the collocation loss L_c. Although at the end of the second stage the total loss is still high, there is a decreasing trend and a considerable reduction during this stage.

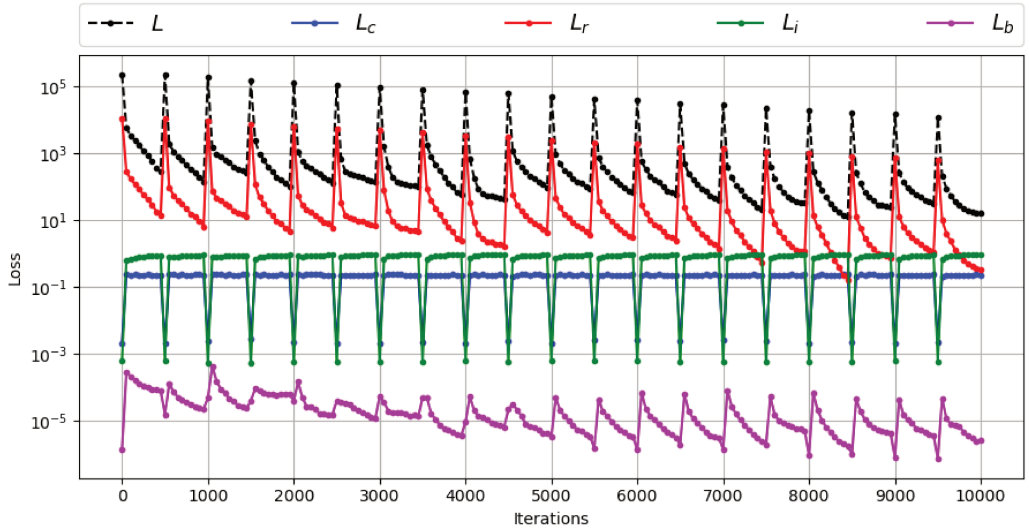

Figure 4.7 – Learning curve for the second optimization stage.

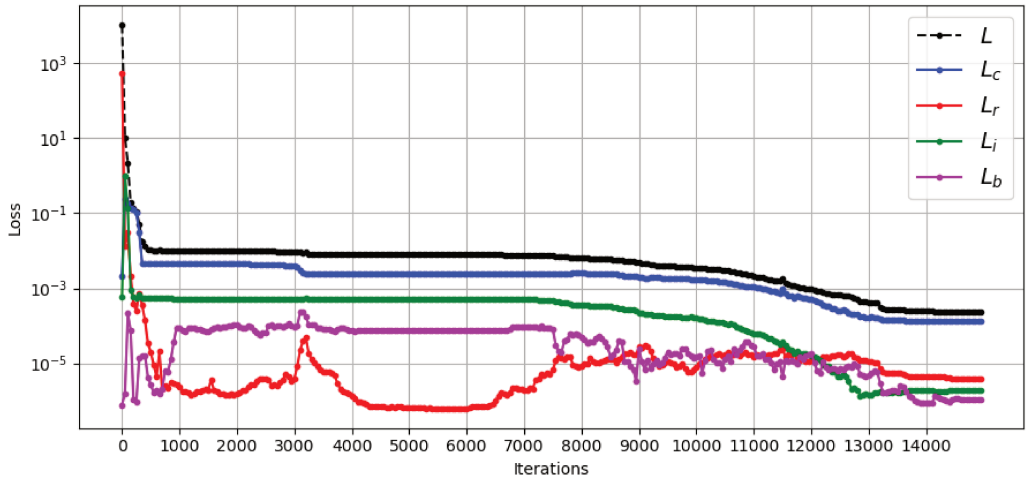

Figure 4.8 – Learning curve for the third optimization stage.

Fig. 4.8 presents the last training stage of the proposed methodology. All the curves fall in the initial iterations, but after this drop, most of them remain fairly unchanged in a long region. The curves decline at the end of the optimization followed by a short plateau, where the method reaches the stopping criterion. The total loss is driven mainly by the behavior of the collocation loss and the boundary, initial and residue losses end around the same values.

4.2.1 PDE constrained optimization in FEniCS

As part of verification and validation, an identification in the FEniCS system was implemented using the same considerations previously stated. This framework has an intuitive mathematical interface, express problem solution using a high-level syntax and maybe its most important feature is that its code generation technology generates parallel optimized low-level C++ code for the solution of forward and adjoint systems. As a consequence of this optimization in the code, it is possible to compute functional and gradient information easily and pass this information to the optimization algorithm.

The computation of the gradient using functional perturbations, finite differences, and other approximation methods are affected by the propagation of errors or expensive computations. Alternatively, the adjoint method computes the gradient of a scalar function with a cheaper procedure similarly to automatic differentiation and requires only one PDE evaluation. As presented in Funke and Farrell (2013), the user describes the forward model, the control parameters and the objective function using a high-level syntax called UFL. The optimization framework then repeatedly re-executes the tape (record of operations used in the solution of the equations) to evaluate the functional value, solves the adjoint PDE to compute the functional gradient, and modifies the tape to update the control parameters until an optimal solution is found. Details about how the adjoint equation is derived using the first-order optimality conditions for a PDE constrained optimization problem can be found in Funke and Farrell (2013).

The identification with FEniCS adjoint employs the damage evolution in terms of parameters λ_1^*, λ_2^* and λ_3^* as

$$\frac{\partial \varphi}{\partial t} = \lambda_1^* \frac{\partial}{\partial x}\Big(\frac{\partial \varphi}{\partial x}\Big) + \lambda_2^*(1-\varphi) - \lambda_3^* \varphi, \tag{4.3}$$

where,

$$\lambda_1^* = \frac{\lambda_1 \lambda_3}{\lambda_2} \tag{4.4}$$

$$\lambda_2^* = \frac{E^*}{\lambda_2} \tag{4.5}$$

$$\lambda_3^* = \frac{\lambda_1}{\lambda_3 \lambda_2}, \tag{4.6}$$

and the results of the estimation are presented in Table 4.4 using $\lambda_1, \lambda_2, \lambda_3$ for comparison. The running time was about 15 minutes in a computer with processor Intel(R) Core(TM) i5-8300H CPU with memory of 12GB RAM.

Table 4.4 – Percentage error in the identification using FEniCS for a bar with constant strain.

Parameter	Label value	Estimated value	Percentage error
λ_1	3.90×10^3	3.89×10^3	0.025%
λ_2	5.00×10^5	4.99×10^5	0.020%
λ_3	6.00×10^{-2}	5.99×10^{-2}	0.017%

In this case, the pseudo-experimental data was obtained using a mesh with 1000 nodes and the functional was constructed with samples in the time domain and considering all the spatial values for a given sample. 5 time steps of a total of 1527 were randomly selected and used to construct the objective function that was minimized.

The identification using this framework gave significant smaller errors for the material parameters in comparison with the PINN methodology. However, the automated PDE constrained optimization required all the spatial information for each time step sample used in the estimation and was performed in terms of linear parameters $\lambda_1^*, \lambda_2^*, \lambda_3^*$. In this respect, the neural network approach is more general and can be easily adapted to experimental settings where all the spatial information is not available. Another important remark is that the use of function approximators has proven to be better to estimate non-linear parameters in differential equation models and also has good performance in the presence of noise. On the other hand, in solutions with classical numerical methods, such as the finite element method, the number of material parameters increases with the refining of the mesh in non-linear problems and some numerical issues can be developed in the presence of noisy data (BERG; NYSTRöM, 2017).

4.2.2 Noise robustness of the methods

One of the advantages of neural networks as function approximators is that they have shown to be robust in the presence of noise (RAISSI; PERDIKARIS; KARNIADAKIS, 2017b; BORODINOV et al., 2019). To asses this, we perform the identification of the parameters using 20 different values of uncorrelated noise between 0 and 10% similar to the systematic study presented in Raissi, Perdikaris and Karniadakis (2017b). The parameters estimated for different levels of noise are in Table 4.5 and Table 4.6 presents the mean, the standard deviation and the relative standard deviation of the identification to evaluate the robustness of the implementation to the noise.

Table 4.5 – Parameters estimated for different levels of noise using a PINN.

Level of noise %	λ_1	λ_2	λ_3
0.0	3772.955	501054.931	0.058
0.5	3913.628	501067.868	0.060
1.0	3757.971	503823.802	0.065
1.5	3771.606	500928.572	0.058
2.0	3663.080	503543.274	0.063
2.5	3765.409	503434.252	0.065
3.0	3861.293	500984.546	0.059
3.5	3805.192	503100.513	0.064
4.0	3856.346	502600.354	0.063
4.5	4116.996	500612.807	0.061
5.0	3855.899	502553.604	0.063
5.5	3625.063	500640.227	0.054
6.0	3891.068	502382.433	0.063
6.5	3596.138	500395.233	0.053
7.0	4186.175	500603.370	0.063
7.5	3520.839	500572.652	0.053
8.0	3913.673	502100.357	0.063
8.5	3866.172	502254.003	0.063
9.0	3945.199	501774.604	0.062
9.5	3971.563	500254.033	0.058
10.0	3453.295	500429.881	0.051

Table 4.6 – Influence of noise in the estimation of the parameters for a PINN.

Parameter	Label value	Mean value	Relative standard deviation	Mean error
λ_1	3.90×10^3	3.81×10^3	4.57%	3.77%
λ_2	5.00×10^5	5.02×10^5	0.23%	0.33%
λ_3	6.00×10^{-2}	6.01×10^{-2}	7.06%	6.02%

We let the same settings to all the levels of noise but we know that it could exist a better combination of hyperparameters for each of them. From the quantitative results in Table 4.6 we see that the estimation of the parameter λ_2 is very accurate and the results for the other parameters are within an acceptable range.

We applied the same test of robustness for the solution using constrained optimization in FEniCS and the results are shown in Tables 4.7 and 4.8. Despite having some good estimates for mean values, we observe that the relative standard deviation and the mean error are higher in contrast to the results of the neural network methodology.

Table 4.7 – Influence of noise in the estimation of the parameters for constrained optimization.

Parameter	Label value	Mean value	Relative standard deviation	Mean error
λ_1	3.90×10^3	3.25×10^3	12.82%	16.66%
λ_2	5.00×10^5	5.01×10^5	0.05%	0.08%
λ_3	6.00×10^{-2}	5.02×10^{-2}	12.58%	16.34%

Table 4.8 – Parameters estimated for different levels of noise in FEniCS

Level of noise	λ_1	λ_2	λ_3
0.0 %	3899.963	499999.990	0.060
0.5 %	3841.518	500037.703	0.059
1.0 %	3782.259	500075.429	0.058
1.5 %	3722.152	500113.166	0.057
2.0 %	3661.160	500150.915	0.056
2.5 %	3599.243	500188.677	0.055
3.0 %	3536.359	500226.450	0.055
3.5 %	3472.442	500264.313	0.054
4.0 %	3407.519	500302.095	0.053
4.5 %	3341.461	500339.888	0.052
5.0 %	3274.204	500377.696	0.051
5.5 %	3205.683	500415.521	0.050
6.0 %	3135.826	500453.362	0.048
6.5 %	3064.554	500491.217	0.047
7.0 %	2991.255	500529.097	0.046
7.5 %	2916.964	500566.998	0.045
8.0 %	2841.060	500604.904	0.044
8.5 %	2763.269	500642.795	0.043
9.0 %	2683.043	500680.681	0.042
9.5 %	2600.532	500718.606	0.040
10.0 %	2515.914	500756.562	0.039

4.3 Identification considering the displacement evolution

After exploring the use of PINNs without considering the displacement evolution, now we return to the coupled system of equations for the identification. In this case, the evolution of displacement is given as an input to the neural network making the inverse analysis more consistent with the real behavior.

We propose a total of four cases with three distinct initial conditions for damage in each of them. The neural network identifies a different set of material parameters in problems with several boundary and initial conditions. With these cases, we evaluate the generalization capabilities and robustness of the implemented methodology.

In all cases, we adopt a bar of length $L = 1\,\mathrm{m}$, cross sectional area $A = 1 \times 10^{-3}\,\mathrm{m}^2$, density $\rho = 2810\,\mathrm{kg\,m}^{-3}$ and Young's modulus $E = 72\,\mathrm{GPa}$. The initial conditions for the displacement equation are $u = 0$ and $\dot{u} = 0$ in the spatial domain.

The pseudo-experimental data used for the identification gives the evolution of damage in its domain and includes the strain distribution in the bar for the learning process. The classical methods used to obtain the training are summarized in Table 4.9

Table 4.9 – Classical numerical solution for the cases with displacement evolution.

Space discretization	Interpolation order - Nodes	Time marching	Δt
FEM	2-1000	α-Method / Newmark	$1 \times 10^{-4}\,\mathrm{s}$

4.3.1 Case 1

In this case, the bar is fixed at the end $x = 0$ and subjected to a displacement $u_L = 1 \times 10^{-3}\,\mathrm{m}$ at the end $x = L$. The three initial conditions tested for damage are presented in Fig. 4.9.

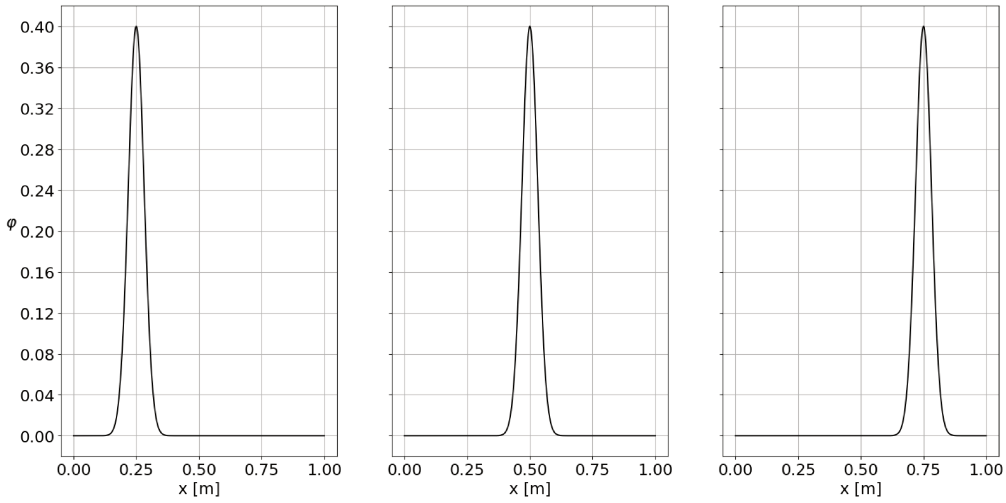

Figure 4.9 – Initial conditions considered for case 1 with $\varphi_0 = 0.4$.

The damage evolution is rewritten in terms of λ_1, λ_2 and λ_3 as

$$\frac{\partial \varphi}{\partial t} = \left(\frac{\lambda_1 \lambda_3}{\lambda_2}\right) \frac{\partial}{\partial x}\left(\frac{\partial \varphi}{\partial x}\right) + \left(\frac{E}{\lambda_2}\right)(1-\varphi)\left(\frac{\partial u}{\partial x}\right)^2 - \left(\frac{\lambda_1}{\lambda_2 \lambda_3}\right)\varphi. \tag{4.7}$$

with the strain calculated from the solutions of the displacement equation

$$\rho \frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\Big((1-\varphi)^2 E \frac{\partial u}{\partial x}\Big). \tag{4.8}$$

The use of both state variables u and φ required some minor changes in the depth and width of the neural network. The hyperparameters adopted for the neural network are presented in Table 4.10, the penalizing weights used in L are $\alpha_r = 10$, $\alpha_i = 8$, $\alpha_b = 2$, $\alpha_c = 10$, and the same configuration is used for the three initial conditions.

Table 4.10 – Hyperparameters of the PINN for case 1.

Layers and neurons	Samples	L-BFGS	Batch Size	Adam + L-BFGS	L-BFGS
[2,25,25,25,25,1]	5000	1000	2000	10000-80(500)	20000

The estimation of the material parameters was performed using an exponential scale and defining bounds with enough range for the parameter search. The percentage errors for the parameters are given in Table 4.11 and the PINN approximation is presented in Figs. 4.10, 4.11 and 4.12.

Table 4.11 – Percentage errors in the identification for case 1.

	Parameter	Label value	Estimated value	Percentage error
φ_0^{max} at $x_0 = 0.25$	λ_1	4.00×10^3	3.76×10^3	5.95%
	λ_2	1.60×10^5	1.69×10^5	5.40%
	λ_3	2.00×10^{-2}	2.14×10^{-2}	6.93%
φ_0^{max} at $x_0 = 0.5$	λ_1	4.00×10^3	3.87×10^3	3.25%
	λ_2	1.60×10^5	1.64×10^5	2.41%
	λ_3	2.00×10^{-2}	2.06×10^{-2}	2.88%
φ_0^{max} at $x_0 = 0.75$	λ_1	4.00×10^3	3.78×10^3	5.50%
	λ_2	1.60×10^5	1.67×10^5	4.26%
	λ_3	2.00×10^{-2}	2.09×10^{-2}	4.48%

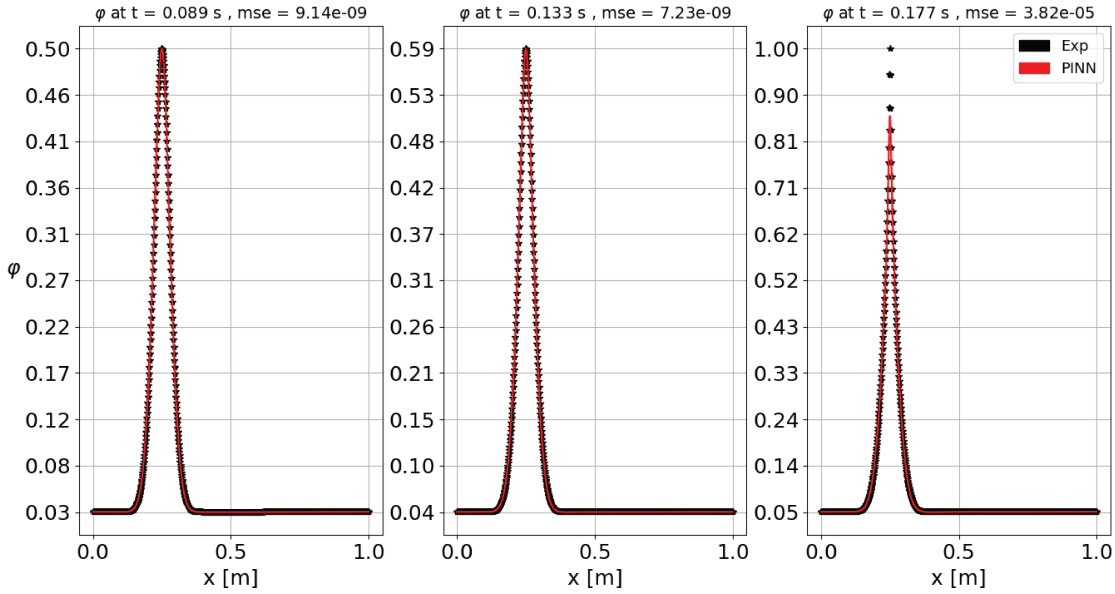

Figure 4.10 – Case 1 with initial condition φ_0^{max} *at* $x_0 = 0.25$.

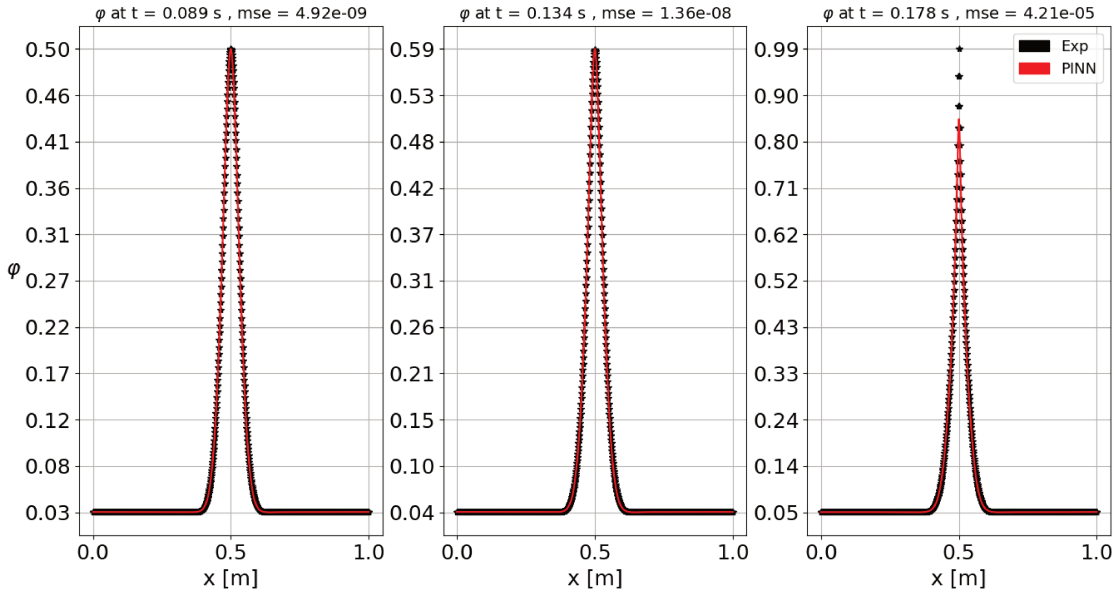

Figure 4.11 – Case 1 with initial condition φ_0^{max} *at* $x_0 = 0.5$.

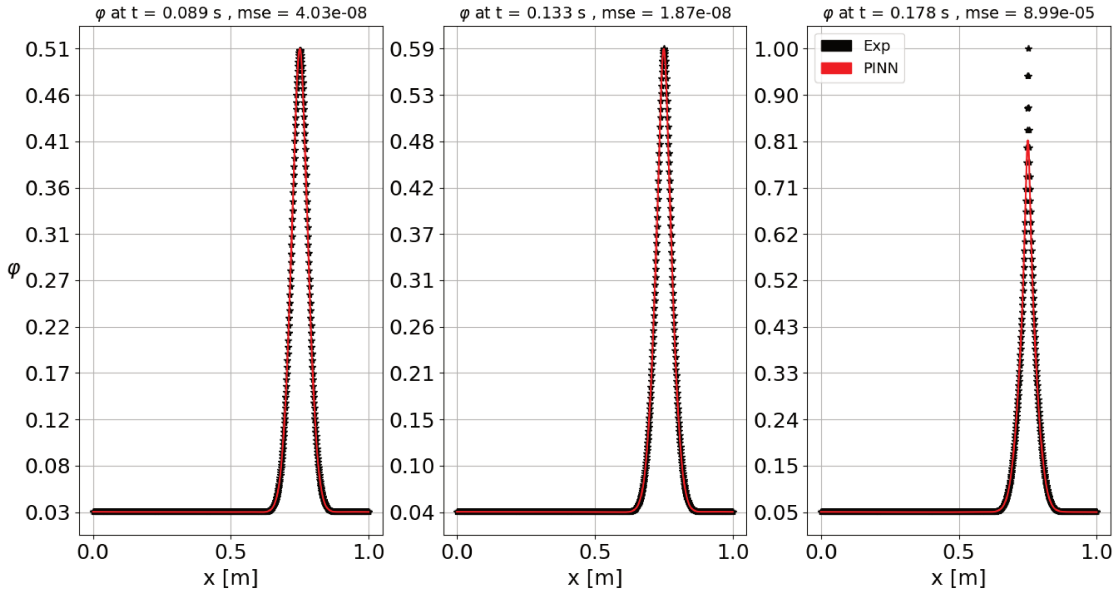

Figure 4.12 – Case 1 with initial condition φ_0^{max} *at* $x_0 = 0.75$.

The material parameters estimated using the physics informed neural network methodology provided good results with percentage errors below a 7%. In general, the errors in the identification were smaller for the parameter λ_2 and had similar values for the others. It is also important to remark that the same configuration was used for all the initial conditions but it is possible to optimize each of them if a smaller error is desired.

In Figs. 4.10 to 4.12, we observe that for all the initial conditions proposed the damage has a similar behavior. The differences between the target values and the output of the neural network are measured using the mean square error (mse) and the PINN solution follows the expected evolution in the figures presented. However, for values close to the final time step of the experimental data the output of the neural network seems to be delayed. This can be a result of a smoothed solution of the neural networks in this closed region.

4.3.2 Case 2

In this case, the bar is fixed at the end $x = 0$ and subjected to a force $F_L = 100\,\mathrm{kN}$ at the end $x = L$. The initial conditions tested for the damage are applied using a Gaussian distribution centered at $x = 0.25$, $x = 0.5$ and $x = 0.75$ with $\varphi_0 = 0.3$. Although this case is physically equivalent to the previous, it considers a Neuman boundary condition for the displacement equation and a different φ_0 for the initial damage.

The hyperparameters adopted for the neural network are presented in Table 4.12. The first row has the configurations for the initial conditions centered at $x = 0.25$ and $x = 0.5$. It was necessary to increase the amount of neurons for the initial condition centered at $x = 0.75$ as presented in the second row. For this and the following two cases, the penalizing weights used in the total loss function are $\alpha_r = 20$, $\alpha_i = 8$, $\alpha_b = 2$, $\alpha_c = 15$.

Table 4.12 – Hyperparameters of the PINN for case 2.

Layers and neurons	Samples	L-BFGS	Batch Size	Adam + L-BFGS	L-BFGS
[2,14,14,14,14,14,1]	8000	1000	2000	10000-80(500)	20000
[2,18,18,18,18,18,1]	8000	1000	2000	10000-80(500)	20000

The percentage errors of the parameter identification are presented in Table 4.13. For this physical case, we had a larger error in λ_3 and a smaller error in λ_1 for all the initial conditions proposed. This could be associated with the sensitivity of the damage response to these constants. In contrast to the previous case, we employed more neurons for the last initial condition considered to have an acceptable percentage error for λ_3.

Table 4.13 – Percentage error in the identification for case 2.

	Parameter	Label value	Estimated value	Percentage error
φ_0^{max} at $x_0 = 0.25$	λ_1	8.00×10^3	7.99×10^3	0.06%
	λ_2	2.00×10^5	2.09×10^5	4.50%
	λ_3	1.00×10^{-2}	1.06×10^{-2}	6.40%
φ_0^{max} at $x_0 = 0.5$	λ_1	8.00×10^3	7.94×10^3	0.72%
	λ_2	2.00×10^5	2.10×10^5	4.82%
	λ_3	1.00×10^{-2}	1.06×10^{-2}	6.29%
φ_0^{max} at $x_0 = 0.75$	λ_1	8.00×10^3	8.17×10^3	2.08%
	λ_2	2.00×10^5	2.10×10^5	5.18%
	λ_3	1.00×10^{-2}	1.10×10^{-2}	9.66%

In Figs. 4.13, 4.14 and 4.15, we perceive the same smoothing effect for the last time step of damage evolution and only slight differences in previous steps for all the initial conditions considered.

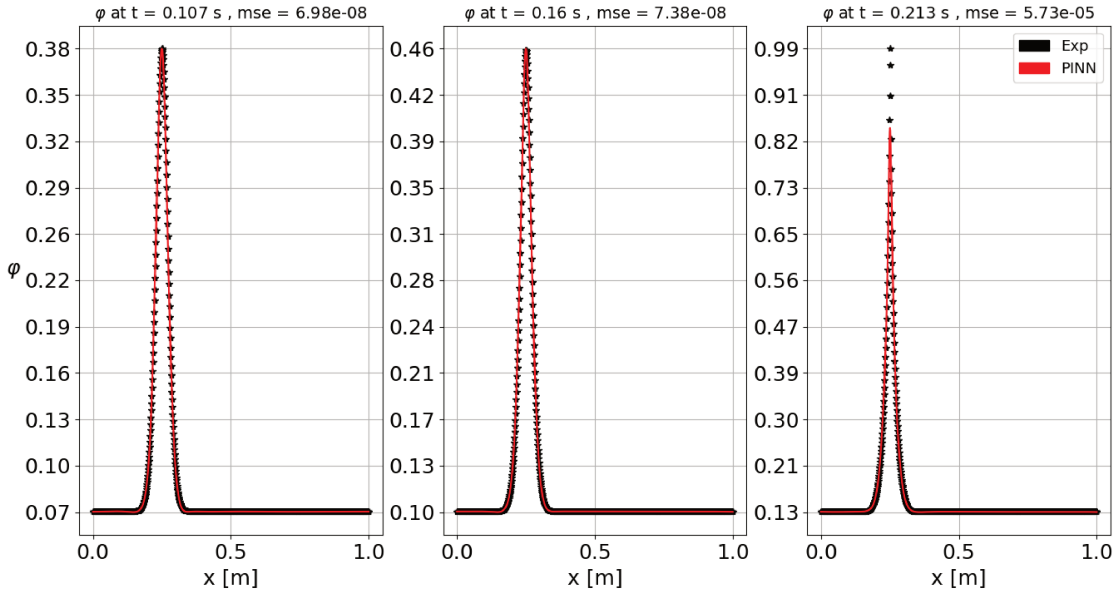

Figure 4.13 – Case 2 : Initial condition φ_0^{max} *at* $x_0 = 0.25$.

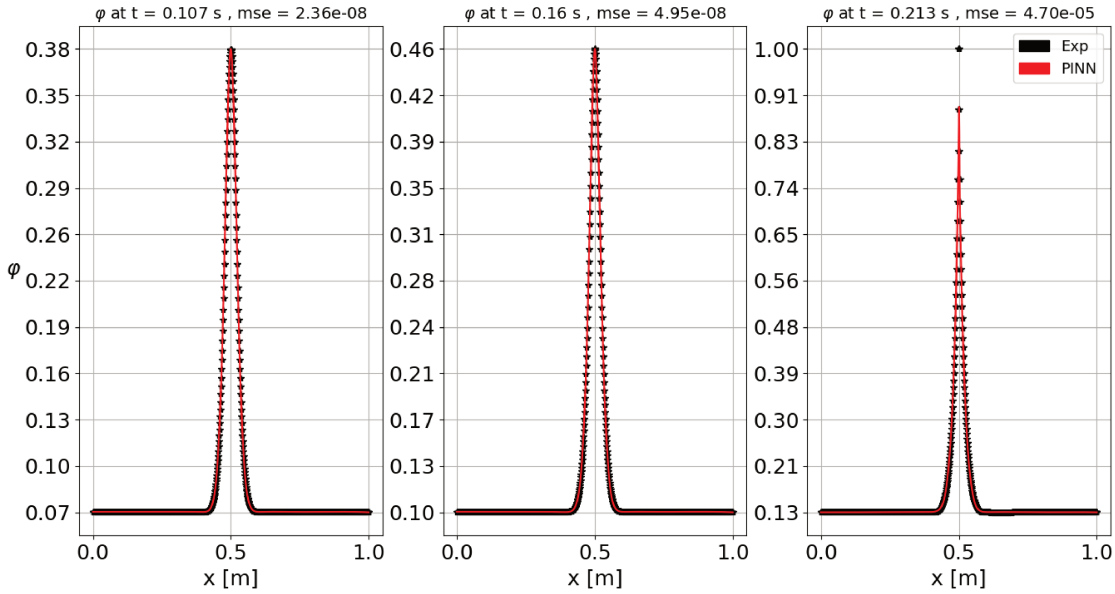

Figure 4.14 – Case 2 : Initial condition φ_0^{max} *at* $x_0 = 0.5$.

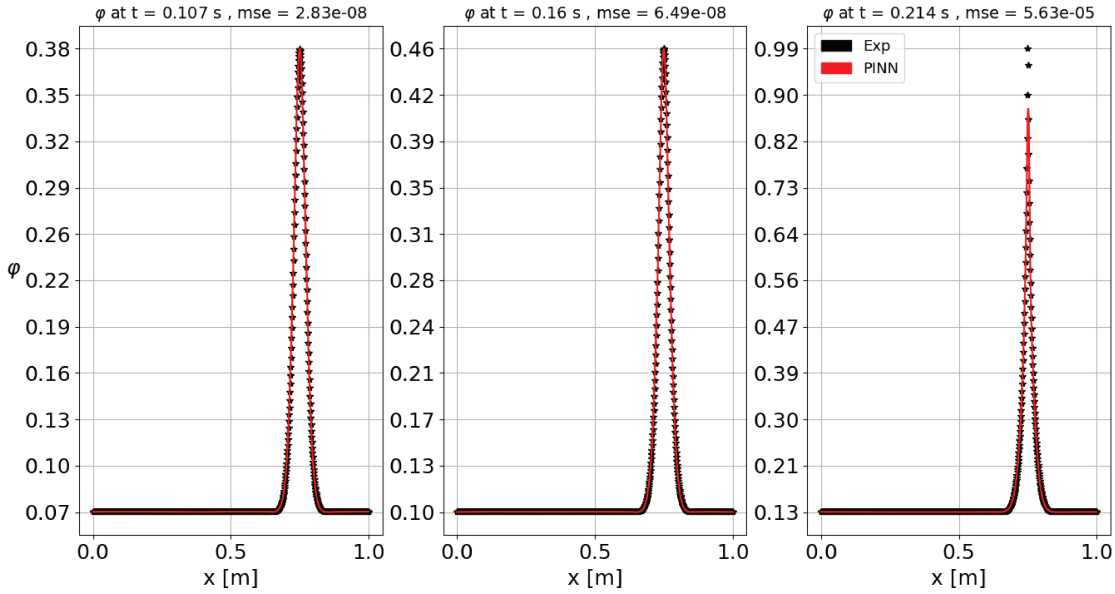

Figure 4.15 – Case 2 : Initial condition φ_0^{max} *at* $x_0 = 0.75$.

4.3.3 Case 3

In this case, the bar is fixed at the end $x = 0$, subjected to a displacement $u_L = 1 \times 10^{-3}\,\mathrm{m}$ at the end $x = L$ and has a constant distributed load $f_r = 150\,\mathrm{kN\,m^{-1}}$. For this and the next case, the strain is calculated from the solutions of the displacement equation

$$\rho \frac{\partial^2 u}{\partial t^2} = \frac{\partial}{\partial x}\Big((1-\varphi)^2 E \frac{\partial u}{\partial x} +\Big) + f_r(x). \tag{4.9}$$

The initial conditions tested for damage are applied using a Gaussian distribution centered at $x = 0.25$, $x = 0.5$ and $x = 0.75$ with $\varphi_0 = 0.25$.

We used the same hyperparameters of the last case and the material parameters were estimated with an acceptable percentage error for the initial conditions centered at $x = 0.5$ and $x = 0.75$. The initial condition centered at $x = 0.25$ required a larger amount of neurons as showed in Table 4.14.

Table 4.14 – Hyperparameters of the PINN for case 3.

Layers and neurons	Samples	L-BFGS	Batch Size	Adam + L-BFGS	L-BFGS
[2,20,20,20,20,20,1]	8000	1000	2000	10000-80(500)	20000

The percentage error of the parameter identification is shown in Table 4.15. In this case, we had a larger error in λ_3 and a smaller error in λ_2 for all the initial conditions proposed. Like in the previous case, we employed more neurons for one of the initial conditions considered. This was necessary to have an acceptable percentage error for the material parameters.

Table 4.15 – Percentage errors in the identification for case 3.

	Parameter	Label value	Estimated value	Percentage error
φ_0^{max} at $x_0 = 0.25$	λ_1	6.00×10^3	5.76×10^3	4.03%
	λ_2	1.70×10^5	1.75×10^5	2.77%
	λ_3	2.00×10^{-2}	2.14×10^{-2}	7.06%
φ_0^{max} at $x_0 = 0.5$	λ_1	6.00×10^3	6.04×10^3	0.71%
	λ_2	1.70×10^5	1.72×10^5	0.97%
	λ_3	2.00×10^{-2}	2.08×10^{-2}	4.07%
φ_0^{max} at $x_0 = 0.75$	λ_1	6.00×10^3	6.22×10^3	3.69%
	λ_2	1.70×10^5	1.72×10^5	1.14%
	λ_3	2.00×10^{-2}	2.17×10^{-2}	8.48%

In Figs. 4.16 to 4.18, it can be seen that although there are significant variations in the damage behavior depending on the initial conditions proposed, the neural network recovers the pseudo-experimental data in each of them. In contrast with previous cases, the differences of the neural network in the last time steps for initial conditions centered around $x = 0.5$ and $x = 0.75$ can not be directly attributed to an smoothing effect. However, we see how the values of damage change rapidly in an small Δx which is something that also happened before and after the peak of the previous cases.

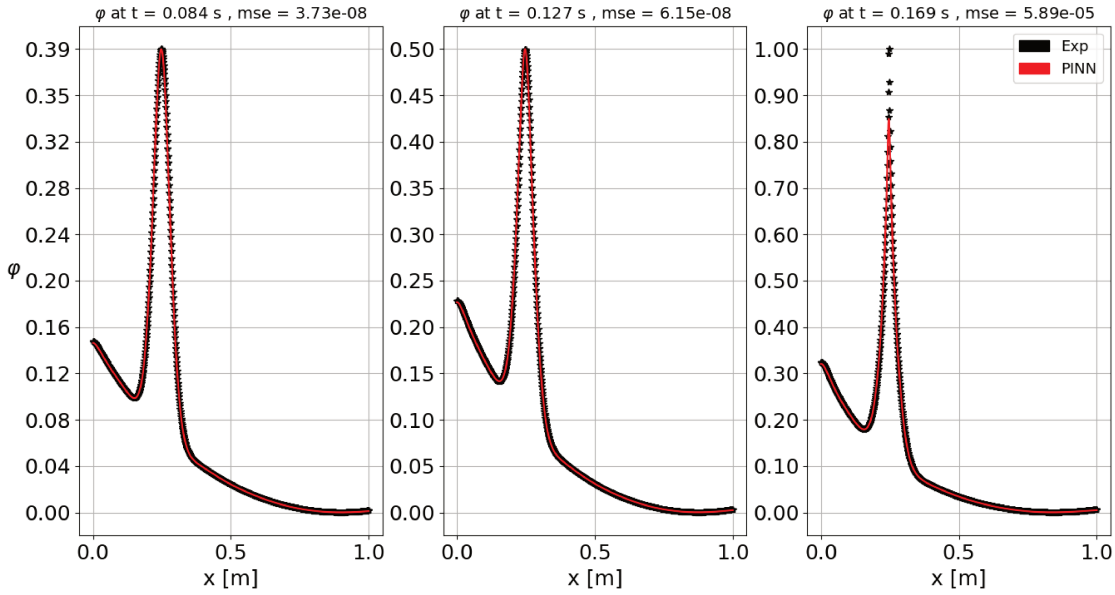

Figure 4.16 – Case 3 : Initial condition φ_0^{max} at $x_0 = 0.25$.

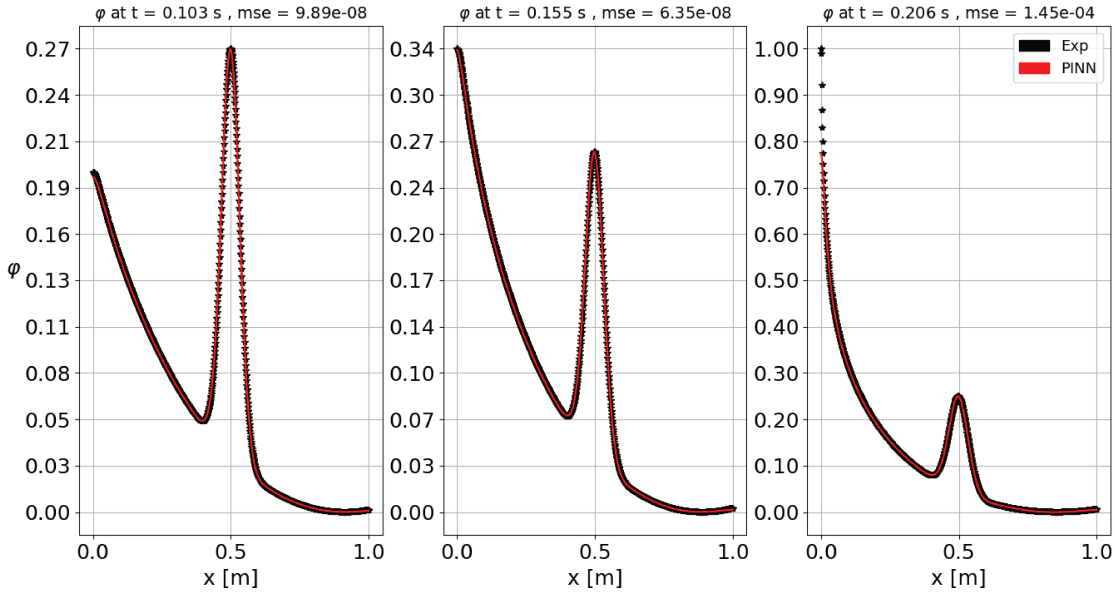

Figure 4.17 – Case 3 : Initial condition φ_0^{max} *at* $x_0 = 0.5$.

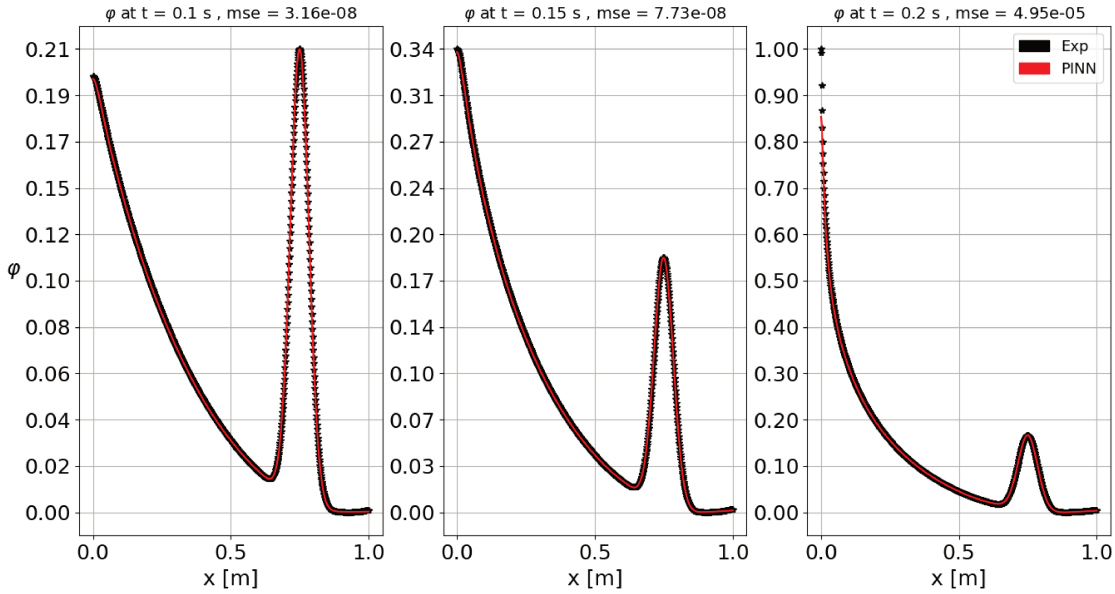

Figure 4.18 – Case 3 : Initial condition φ_0^{max} *at* $x_0 = 0.75$.

4.3.4 Case 4

In this case, the bar is fixed at the end $x = 0$, subjected to a displacement $u_L = 1 \times 10^{-3}\,\mathrm{m}$ at the end $x = L$ and has a distributed load $f_r = 150 \sin\left(\frac{\pi x}{2}\right)\,\mathrm{kN\,m^{-1}}$. The three

initial conditions tested for damage are applied using a Gaussian distribution centered at $x = 0.25$, $x = 0.5$ and $x = 0.75$ with $\varphi_0 = 0.35$.

For this case, we use a neural network with 6 hidden layers of 14 neurons for the initial conditions centered at $x = 0.25$ and $x = 0.75$ and an architecture with 5 hidden layers and the same number of neurons for the initial condition centered at $x = 0.5$. All the configurations of the optimization stages are the same of the previous cases.

The percentage errors of the parameter identification are given in Table 4.16. In this case, we had a larger error in λ_3 for all the initial conditions proposed. Similarly to the previous cases (2 and 3), we needed a slightly different configuration for two of the initial conditions considered.

Table 4.16 – Percentage error in the identification for case 4.

	Parameter	Label value	Estimated value	Percentage error
φ_0^{max} at $x_0 = 0.25$	λ_1	6.00×10^3	5.57×10^3	7.16%
	λ_2	1.70×10^5	1.82×10^5	7.11%
	λ_3	2.00×10^{-2}	2.22×10^{-2}	11.25%
φ_0^{max} at $x_0 = 0.5$	λ_1	6.00×10^3	6.02×10^3	0.33%
	λ_2	1.70×10^5	1.71×10^5	0.68%
	λ_3	2.00×10^{-2}	2.04×10^{-2}	1.81%
φ_0^{max} at $x_0 = 0.75$	λ_1	6.00×10^3	6.16×10^3	2.65%
	λ_2	1.70×10^5	1.71×10^5	0.66%
	λ_3	2.00×10^{-2}	2.09×10^{-2}	4.57%

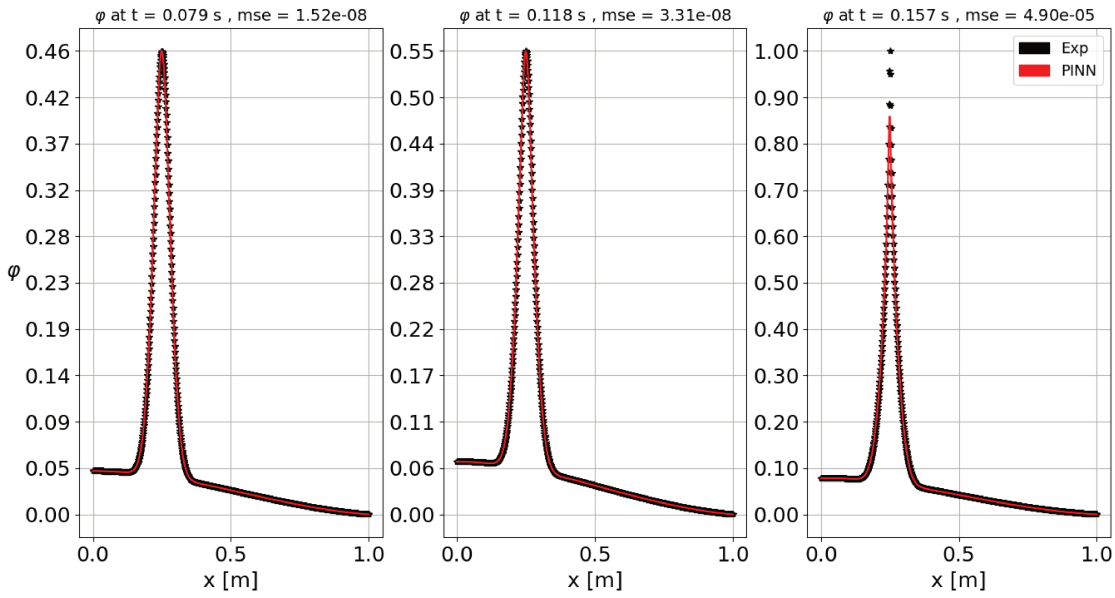

Figure 4.19 – Case 4 : Initial condition φ_0^{max} at $x_0 = 0.25$.

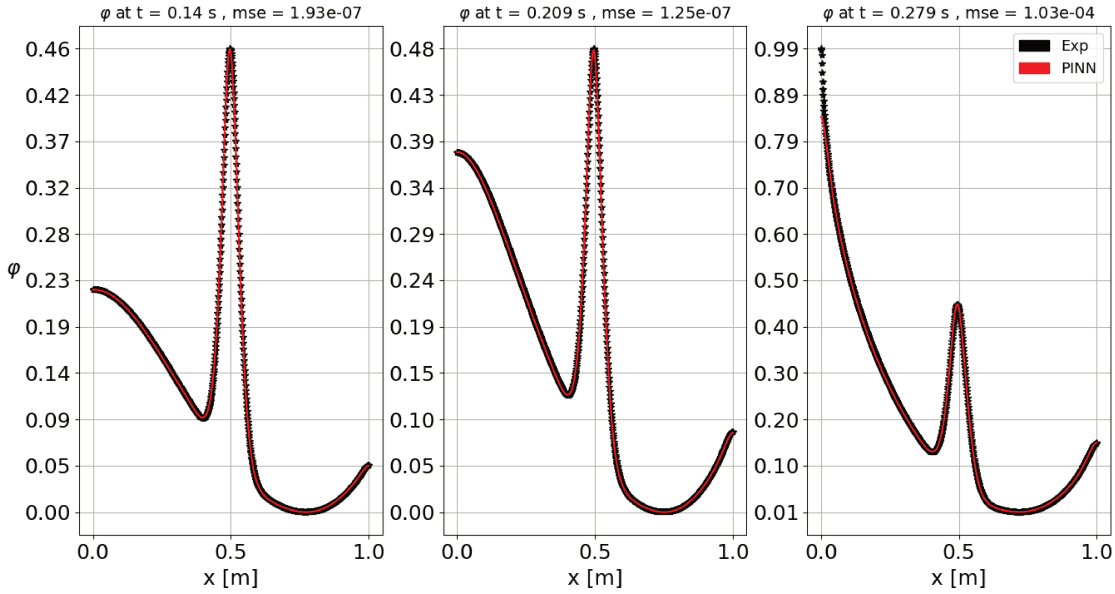

Figure 4.20 – Case 4 : Initial condition φ_0^{max} *at* $x_0 = 0.5$.

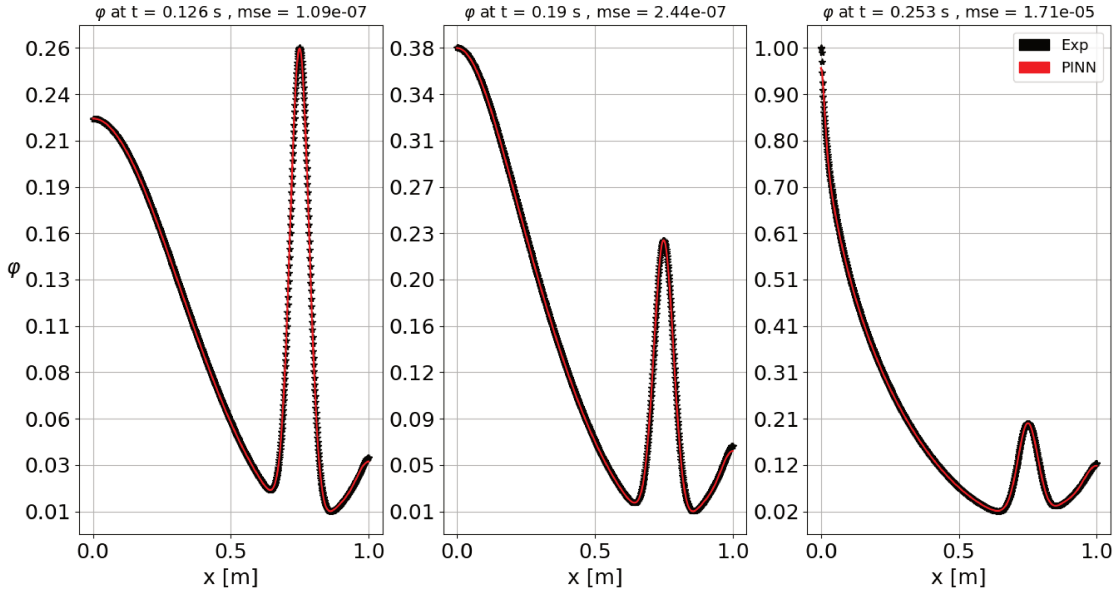

Figure 4.21 – Case 4 : Initial condition φ_0^{max} *at* $x_0 = 0.75$.

In Figs. 4.19 to 4.21, we see that, as in the previous case, there are significant variations in the damage evolution depending on the initial conditions proposed. Nevertheless, the neural network recovered well the pseudo-experimental data in each of them. It can be noticed that the approximation in the last step is better for the initial condition centered around $x = 0.75$ than $x = 0.25$. This shows that the hyperparameters selected are better in some cases and depending on the conditions is better to adopt particular considerations for each physical case.

5 Conclusions and suggestions for future research

This work addressed the estimation of parameters in the governing equations of the damage model proposed in Boldrini et al. (2016). We implemented the identification of three material parameters applying a physics informed neural network and combining some ideas of the two-step method, the principal differential analysis, and the generalized smoothing approach. Initially, the identification strategy was tested using only the evolution of the damage and afterward, it was included an additional input in the neural network model with information from the solution of the displacement equation. We examined the robustness of the method in the presence of noisy training data and also their generalization capabilities in different physical cases.

At the beginning of the dissertation, we presented a literature review with works that have applied neural networks in the solution of differential equations and also in the estimation of parameters in models. Then we introduced the formulation of the material identification as an optimization problem and described some popular techniques that have been used over the years. There are different levels of classification for the methods to solve this type of inverse problem, but it seems that exists a trend towards the adoption of function approximators to fit the models using Bayesian inference. These choices are popular because they work well in the presence of noise and also can introduce prior information in the solution to the problem. In this work, we used a neural network as function approximator and a deterministic approach to estimate the parameters.

We adopted the physics informed methodology as the base for our neural network model because it has been proved in different identification problems with good results. However, we decided to make some modifications in the formulation of the optimization process because we observed that when the term that fits the observations and the residue of the governing equation of the model have the same importance, the neural network demands a higher number of neurons and layers. We also now from the work of Ramsay et al. (2007) that the simultaneous search for the parameters of the neural network, i.e., nuisance parameters, can complicate the estimation of the material parameters, i.e., structural parameters. In order to avoid this and local minima trouble, we proposed three stages of optimization. First, we optimize the collocation loss which means that in this stage only the parameters of the network are refined. In the second stage, we alternate between a simultaneous search of all the parameters using a gradient descent method, and the optimization of only the neural network parameters using an L-BFGS algorithm. Finally, we complemented the estimation of the parameters implementing a simultaneous search with a high execution limit.

The equations of the model for the hypothesis presented in Chapter 2 give the evolution

of the displacement and damage and were applied in a simple physical case. We used a bar with the left extreme fixed and diverse boundary conditions for the right end. With the purpose of exploring different approaches, we simplified the identification problem considering only the damage equation with a constant strain in the bar. We tested the robustness of the implementation using various levels of noise in the training data. Although the mean errors for the different levels of noise were larger than those of the clean data, the quantitative results were within an acceptable range considering that we kept the same hyperparameters tuned for the original (clean) training data. In addition, we presented the results of the estimation using a method that constrains the optimization using the numerical solution of the differential equation. Despite that the results for the clean data were superior, we found that the mean error in the presence of noise was higher for the constrained method. In this case, the neural network methodology had better performance when dealing with noisy data which is a property desired to work with experimental observations.

Finally, we proposed four physical cases to evaluate the generalization capabilities of the strategy proposed. These cases had different boundary conditions, initial conditions and some included distributed loads. We used a random search to tune one of the configurations in each case and only introduced minor changes in the number of neurons and layers to maintain the error controlled when it was necessary. In general, just slight modifications were needed and the quantitative and qualitative results were satisfactory.

5.1 Suggestions for future works

Some possible ideas for future works to continue with this line of research are summarized in the following list:

- Propose other physical cases for the application of the damage model and generalize the estimation process using directly the terms from the free energy and the pseudo-potential of dissipation as physics regularization.
- Consider the evolution of the displacement using an additional neural network or a new output in the same network.
- Explore the use of a global optimization algorithm to avoid local minima solutions and poor approximations. An interesting alternative could be to create a surrogate convex loss to improve the performance of local optimization methods without increasing the computational cost.

- Refine the selection of the collocation points using information from the loss during the training process. This can be achieved using another machine learning system.
- Implement cross-validation to select the penalization weights of the terms in the loss function.

References

AARTS, L. P.; VEER, P. van der. **Neural Processing Letters**, Springer Science and Business Media LLC, v. 14, n. 3, p. 261–271, 2001. Available at: <https://doi.org/10.1023/a:1012784129883>.

ABADI, M.; AGARWAL, A.; BARHAM, P.; BREVDO, E.; CHEN, Z.; CITRO, C.; CORRADO, G. S.; DAVIS, A.; DEAN, J.; DEVIN, M.; GHEMAWAT, S.; GOODFELLOW, I.; HARP, A.; IRVING, G.; ISARD, M.; JIA, Y.; JOZEFOWICZ, R.; KAISER, L.; KUDLUR, M.; LEVENBERG, J.; MANé, D.; MONGA, R.; MOORE, S.; MURRAY, D.; OLAH, C.; SCHUSTER, M.; SHLENS, J.; STEINER, B.; SUTSKEVER, I.; TALWAR, K.; TUCKER, P.; VANHOUCKE, V.; VASUDEVAN, V.; VIéGAS, F.; VINYALS, O.; WARDEN, P.; WATTENBERG, M.; WICKE, M.; YU, Y.; ZHENG, X. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. 2015. Software available from tensorflow.org. Available at: <https://www.tensorflow.org/>.

ABIODUN, O. I.; JANTAN, A.; OMOLARA, A. E.; DADA, K. V.; MOHAMED, N. A.; ARSHAD, H. State-of-the-art in artificial neural network applications: A survey. **Heliyon**, Elsevier BV, v. 4, n. 11, p. e00938, Nov. 2018. Available at: <https://doi.org/10.1016/j.heliyon.2018.e00938>.

ADLER, J.; ÖKTEM, O. Solving ill-posed inverse problems using iterative deep neural networks. **Inverse Problems**, v. 33, n. 12, p. 124007, Dec. 2017. ISSN 0266-5611, 1361-6420. Available at: <https://iopscience.iop.org/article/10.1088/1361-6420/aa9581>.

ALLI, H.; UÇAR, A.; DEMIR, Y. The solutions of vibration control problems using artificial neural networks. **Journal of the Franklin Institute**, Elsevier BV, v. 340, n. 5, p. 307–325, Aug. 2003. Available at: <https://doi.org/10.1016/s0016-0032(03)00036-x>.

ASTER BRIAN BORCHERS, C. H. T. R. C. **Parameter Estimation and Inverse Problems**. Elsevier, 2019. Available at: <https://doi.org/10.1016/c2015-0-02458-3>.

BALAY, S.; ABHYANKAR, S.; ADAMS, M. F.; BROWN, J.; BRUNE, P.; BUSCHELMAN, K.; DALCIN, L.; DENER, A.; EIJKHOUT, V.; GROPP, W. D.; KARPEYEV, D.; KAUSHIK, D.; KNEPLEY, M. G.; MAY, D. A.; MCINNES, L. C.; MILLS, R. T.; MUNSON, T.; RUPP, K.; SANAN, P.; SMITH, B. F.; ZAMPINI, S.; ZHANG, H.; ZHANG, H. **PETSc Web page**. 2019. <https://www.mcs.anl.gov/petsc>. Available at: <https://www.mcs.anl.gov/petsc>.

BAYDIN, A. G.; PEARLMUTTER, B. A.; RADUL, A. A.; SISKIND, J. M. Automatic differentiation in machine learning: a survey. **arXiv e-prints**, p. arXiv:1502.05767, Feb 2015.

BERAHAS, A. S.; JAHANI, M.; TAKáč, M. Quasi-Newton Methods for Deep Learning: Forget the Past, Just Sample. **arXiv e-prints**, p. arXiv:1901.09997, Jan 2019.

BERG, J.; NYSTRöM, K. A unified deep artificial neural network approach to partial differential equations in complex geometries. **Neurocomputing**, Elsevier BV, v. 317, p. 28–41, Nov. 2018. Available at: <https://doi.org/10.1016/j.neucom.2018.06.056>.

BERG, J.; NYSTRöM, K. Neural network augmented inverse problems for PDEs. **arXiv e-prints**, p. arXiv:1712.09685, Dec 2017.

BOLDRINI, J.; MORAES, E. B. de; CHIARELLI, L.; FUMES, F.; BITTENCOURT, M. A non-isothermal thermodynamically consistent phase field framework for structural damage and fatigue. **Computer Methods in Applied Mechanics and Engineering**, Elsevier, v. 312, p. 395–427, 2016.

BORDEN, M. J.; HUGHES, T. J.; LANDIS, C. M.; VERHOOSEL, C. V. A higher-order phase-field model for brittle fracture: Formulation and analysis within the isogeometric analysis framework. **Computer Methods in Applied Mechanics and Engineering**, Elsevier BV, v. 273, p. 100–118, May 2014. Available at: <https://doi.org/10.1016/j.cma.2014.01.016>.

BORODINOV, N.; NEUMAYER, S.; KALININ, S. V.; OVCHINNIKOVA, O. S.; VASUDEVAN, R. K.; JESSE, S. Deep neural networks for understanding noisy data applied to physical property extraction in scanning probe microscopy. **npj Computational Materials**, v. 5, n. 1, p. 25, Dec. 2019. ISSN 2057-3960. Available at: <http://www.nature.com/articles/s41524-019-0148-5>.

BULJAK, V. **Inverse Analyses with Model Reduction**. Springer Berlin Heidelberg, 2012. Available at: <https://doi.org/10.1007/978-3-642-22703-5>.

CAETANO, C.; REIS, J. L.; AMORIM, J.; LEMES, M. R.; PINO, A. D. Using neural networks to solve nonlinear differential equations in atomic and molecular physics. **International Journal of Quantum Chemistry**, Wiley, v. 111, n. 12, p. 2732–2740, May 2010. Available at: <https://doi.org/10.1002/qua.22572>.

CAO, J.; HUANG, J. Z.; WU, H. Penalized Nonlinear Least Squares Estimation of Time-Varying Parameters in Ordinary Differential Equations. **Journal of Computational and Graphical Statistics**, v. 21, n. 1, p. 42–56, Jan. 2012. ISSN 1061-8600, 1537-2715. Available at: <http://www.tandfonline.com/doi/abs/10.1198/jcgs.2011.10021>.

CAO, J.; RAMSAY, J. O. Parameter cascades and profiling in functional data analysis. **Computational Statistics**, v. 22, n. 3, p. 335–351, Aug. 2007. ISSN 0943-4062, 1613-9658. Available at: <http://link.springer.com/10.1007/s00180-007-0044-1>.

CHAKRAVERTY, S.; MALL, S. **Artificial Neural Networks for Engineers and Scientists**. CRC Press, 2017. Available at: <https://doi.org/10.1201/9781315155265>.

CHIARELLI, L.; FUMES, F.; MORAES, E. B. de; HAVEROTH, G.; BOLDRINI, J.; BITTENCOURT, M. Comparison of high order finite element and discontinuous galerkin methods for phase field equations: Application to structural damage. **Computers & Mathematics with Applications**, Elsevier, 2017.

DUA, V. An artificial neural network approximation based decomposition approach for parameter estimation of system of ordinary differential equations. **Computers & Chemical Engineering**, Elsevier BV, v. 35, n. 3, p. 545–553, Mar. 2011. Available at: <https://doi.org/10.1016/j.compchemeng.2010.06.005>.

DUA, V.; DUA, P. A simultaneous approach for parameter estimation of a system of ordinary differential equations, using artificial neural network approximation. **Industrial & Engineering Chemistry Research**, American Chemical Society (ACS), v. 51, n. 4, p. 1809–1814, Sep. 2011. Available at: <https://doi.org/10.1021/ie200617d>.

EFFATI, S.; PAKDAMAN, M. Artificial neural network approach for solving fuzzy differential equations. **Information Sciences**, Elsevier BV, v. 180, n. 8, p. 1434–1457, Apr. 2010. Available at: <https://doi.org/10.1016/j.ins.2009.12.016>.

FILICI, C. Error estimation in the neural network solution of ordinary differential equations. **Neural Networks**, Elsevier BV, v. 23, n. 5, p. 614–617, Jun. 2010. Available at: <https://doi.org/10.1016/j.neunet.2009.05.014>.

FOJDL, J.; BRAUSE, R. W. The performance of approximating ordinary differential equations by neural nets. In: **2008 20th IEEE International Conference on Tools with Artificial Intelligence**. IEEE, 2008. Available at: <https://doi.org/10.1109/ictai.2008.44>.

FRASSO, G.; JAEGER, J.; LAMBERT, P. Parameter estimation and inference in dynamic systems described by linear partial differential equations. **AStA Advances in Statistical Analysis**, v. 100, n. 3, p. 259–287, Jul. 2016. ISSN 1863-8171, 1863-818X. Available at: <http://link.springer.com/10.1007/s10182-015-0257-5>.

FUNKE, S. W.; FARRELL, P. E. A framework for automated PDE-constrained optimisation. **arXiv e-prints**, p. arXiv:1302.3894, Feb 2013.

GHABOUSSI, J. Advances in neural networks in computational mechanics and engineering. In: **Advances of Soft Computing in Engineering**. [S.l.]: Springer, 2010. chap. 4, p. 191–236.

GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: **In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics**. [S.l.: s.n.], 2010.

GOLL, C.; WICK, T.; WOLLNER, W. DOpElib: Differential Equations and Optimization Environment; A Goal Oriented Software Library for Solving PDEs and Optimization Problems with PDEs. **Archive of Numerical Software**, Vol 5, Jul. 2017. Available at: <http://journals.ub.uni-heidelberg.de/index.php/ans/article/view/11815>.

GROHS, P.; HORNUNG, F.; JENTZEN, A.; ZIMMERMANN, P. **Space-time error estimates for deep neural network approximations for differential equations**. 2019.

GULIKERS, T. **An Integrated Machine Learning and Finite Element Analysis Framework, applied to Composite Substructures including Damage**. Master's Thesis (Master's Thesis) — Delft University of Technology, 12 2018.

HAKIM, V.; KARMA, A. Laws of crack motion and phase-field models of fracture. **Journal of the Mechanics and Physics of Solids**, Elsevier BV, v. 57, n. 2, p. 342–368, Feb. 2009. Available at: <https://doi.org/10.1016/j.jmps.2008.10.012>.

HAVEROTH, G. A.; MORAES, E. A.; BOLDRINI, J. L.; BITTENCOURT, M. L. Comparison of semi and fully-implicit time integration schemes applied to a damage and fatigue phase field model. **Latin American Journal of Solids and Structures**, SciELO Brasil, v. 15, n. 5, 2018.

HAYKIN, S. S.; HAYKIN, S. S. **Neural networks and learning machines**. 3rd ed. ed. New York: Prentice Hall, 2009. OCLC: ocn237325326. ISBN 9780131471399.

HE, S.; REIF, K.; UNBEHAUEN, R. Multilayer neural networks for solving a class of partial differential equations. **Neural Networks**, Elsevier BV, v. 13, n. 3, p. 385–396, Apr. 2000. Available at: <https://doi.org/10.1016/s0893-6080(00)00013-7>.

JADAMBA, B.; KHAN, A. A.; SAMA, M. Inverse problems of parameter identification in partial differential equations. In: **Mathematics in Science and Technology**. World scientific, 2011. Available at: <https://doi.org/10.1142/9789814338820_0009>.

LAGARIS, I.; LIKAS, A.; FOTIADIS, D. Artificial neural network methods in quantum mechanics. **Computer Physics Communications**, Elsevier BV, v. 104, n. 1-3, p. 1–14, Aug. 1997. Available at: <https://doi.org/10.1016/s0010-4655(97)00054-4>.

LAGARIS, I.; LIKAS, A.; FOTIADIS, D. Artificial neural networks for solving ordinary and partial differential equations. **IEEE Transactions on Neural Networks**, Institute of Electrical and Electronics Engineers (IEEE), v. 9, n. 5, p. 987–1000, 1998. Available at: <https://doi.org/10.1109/72.712178>.

LEMAITRE, J.; DESMORAT, R. **Engineering damage mechanics: ductile, creep, fatigue and brittle failures**. [S.l.]: Springer Science & Business Media, 2005.

LENAIL, A. NN-SVG: Publication-Ready Neural Network Architecture Schematics. **Journal of Open Source Software**, v. 4, n. 33, p. 747, Jan. 2019. ISSN 2475-9066. Available at: <http://joss.theoj.org/papers/10.21105/joss.00747>.

LI, H.; SCHWAB, J.; ANTHOLZER, S.; HALTMEIER, M. Nett: Solving inverse problems with deep neural networks. **Inverse Problems**, Jan. 2020. ISSN 0266-5611, 1361-6420. Available at: <https://iopscience.iop.org/article/10.1088/1361-6420/ab6d57>.

LI, X.; OUYANG, J.; JIANG, T.; YANG, B. Integration modified wavelet neural networks for solving thin plate bending problem. **Applied Mathematical Modelling**, Elsevier BV, v. 37, n. 5, p. 2983–2994, Mar. 2013. Available at: <https://doi.org/10.1016/j.apm.2012.07.036>.

LONG, Z.; LU, Y.; MA, X.; DONG, B. **PDE-Net: Learning PDEs from Data**. 2017.

MALEK, A.; BEIDOKHTI, R. S. Numerical solution for high order differential equations using a hybrid neural network—optimization method. **Applied Mathematics and Computation**, Elsevier BV, v. 183, n. 1, p. 260–271, Dec. 2006. Available at: <https://doi.org/10.1016/j.amc.2006.05.068>.

MALL, S.; CHAKRAVERTY, S. Regression-based neural network training for the solution of ordinary differential equations. **International Journal of Mathematical Modelling and Numerical Optimisation**, Inderscience Publishers, v. 4, n. 2, p. 136, 2013. Available at: <https://doi.org/10.1504/ijmmno.2013.055203>.

MALL, S.; CHAKRAVERTY, S. Chebyshev neural network based model for solving lane–emden type equations. **Applied Mathematics and Computation**, Elsevier BV, v. 247, p. 100–114, Nov. 2014. Available at: <https://doi.org/10.1016/j.amc.2014.08.085>.

MALL, S.; CHAKRAVERTY, S. Multi layer versus functional link single layer neural network for solving nonlinear singular initial value problems. In: **Proceedings of the Third International Symposium on Women in Computing and Informatics**. ACM Press, 2015. Available at: <https://doi.org/10.1145/2791405.2791542>.

MALL, S.; CHAKRAVERTY, S. Application of legendre neural network for solving ordinary differential equations. **Applied Soft Computing**, Elsevier BV, v. 43, p. 347–356, Jun. 2016. Available at: <https://doi.org/10.1016/j.asoc.2015.10.069>.

MANZHOS, S.; CARRINGTON, T. An improved neural network method for solving the schrödinger equation. **Canadian Journal of Chemistry**, Canadian Science Publishing, v. 87, n. 7, p. 864–871, Jul. 2009. Available at: <https://doi.org/10.1139/v09-025>.

MASMOUDI, N. K.; REKIK, C.; DJEMEL, M.; DERBEL, N. Two coupled neural-networks-based solution of the hamilton–jacobi–bellman equation. **Applied Soft Computing**, Elsevier BV, v. 11, n. 3, p. 2946–2963, Apr. 2011. Available at: <https://doi.org/10.1016/j.asoc.2010.11.015>.

MEADE, A.; FERNANDEZ, A. The numerical solution of linear ordinary differential equations by feedforward neural networks. **Mathematical and Computer Modelling**, Elsevier BV, v. 19, n. 12, p. 1–25, Jun. 1994. Available at: <https://doi.org/10.1016/0895-7177(94)90095-7>.

MEER, R. van der. **Solving Partial Diferential Equations with Neural Networks**. Master's Thesis (Master's Thesis) — Delft University of Technology, 6 2019.

MEHRKANOON, S.; FALCK, T.; SUYKENS, J. A. Parameter estimation for time varying dynamical systems using least squares support vector machines. **IFAC Proceedings Volumes**, Elsevier BV, v. 45, n. 16, p. 1300–1305, Jul. 2012. Available at: <https://doi.org/10.3182/20120711-3-be-2027.00044>.

MENG, X.; KARNIADAKIS, G. E. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. **Journal of Computational Physics**, v. 401, p. 109020, Jan 2020.

MIEHE, C.; HOFACKER, M.; WELSCHINGER, F. A phase field model for rate-independent crack propagation: Robust algorithmic implementation based on operator splits. **Computer Methods in Applied Mechanics and Engineering**, Elsevier BV, v. 199, n. 45-48, p. 2765–2778, Nov. 2010. Available at: <https://doi.org/10.1016/j.cma.2010.04.011>.

MOLES, C. G. Parameter estimation in biochemical pathways: A comparison of global optimization methods. **Genome Research**, Cold Spring Harbor Laboratory, v. 13, n. 11, p. 2467–2474, Nov. 2003. Available at: <https://doi.org/10.1101/gr.1262503>.

MüLLER, T. G.; TIMMER, J. PARAMETER IDENTIFICATION TECHNIQUES FOR PARTIAL DIFFERENTIAL EQUATIONS. **International Journal of Bifurcation and Chaos**, v. 14, n. 06, p. 2053–2060, Jun. 2004. ISSN 0218-1274, 1793-6551. Available at: <https://www.worldscientific.com/doi/abs/10.1142/S0218127404010424>.

NIELSEN, M. **Neural networks and deep learning**. Determination Press, 2015. Available at: <http://static.latexstudio.net/article/2018/0912/neuralnetworksanddeeplearning.pdf>.

NOCEDAL, J.; WRIGHT, S. J. **Numerical optimization**. New York, N.Y: Springer, 2006. OCLC: 901634931. ISBN 9780387400655 9780387303031.

POYTON, A.; VARZIRI, M.; MCAULEY, K.; MCLELLAN, P.; RAMSAY, J. Parameter estimation in continuous-time dynamic models using principal differential analysis. **Computers**

& Chemical Engineering, v. 30, n. 4, p. 698–708, Feb. 2006. ISSN 00981354. Available at: <https://linkinghub.elsevier.com/retrieve/pii/S0098135405003042>.

RAGAB S. A., F. H. E. **Introduction to Finite Element Analysis for Engineers**. [S.l.]: CRC Press, 2018.

RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. E. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. **arXiv e-prints**, p. arXiv:1711.10561, Nov 2017.

RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. E. Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations. **arXiv e-prints**, p. arXiv:1711.10566, Nov 2017.

RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. E. **Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems**. 2018.

RAISSI, M.; RAMEZANI, N.; SESHAIYER, P. On parameter estimation approaches for predicting disease transmission through optimization, deep learning and statistical inference methods. **Letters in Biomathematics**, Informa UK Limited, p. 1–26, Oct. 2019. Available at: <https://doi.org/10.1080/23737867.2019.1676172>.

RAISSI, M.; YAZDANI, A.; KARNIADAKIS, G. E. **Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data**. 2018.

RAMSAY, J. O.; HOOKER, G.; CAMPBELL, D.; CAO, J. Parameter estimation for differential equations: a generalized smoothing approach: Parameter Estimation for Differential Equations. **Journal of the Royal Statistical Society: Series B (Statistical Methodology)**, v. 69, n. 5, p. 741–796, Nov. 2007. ISSN 13697412. Available at: <http://doi.wiley.com/10.1111/j.1467-9868.2007.00610.x>.

RAMSUNDAR, B.; ZADEH, R. B. **TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning**. 1st. ed. [S.l.]: O'Reilly Media, Inc., 2018. ISBN 1491980451.

ROBITAILLE, B.; MARCOS, B.; VEILLETTE, M.; PAYRE, G. Modified quasi-newton methods for training neural networks. **Computers & Chemical Engineering**, Elsevier BV, v. 20, n. 9, p. 1133–1140, Sep. 1996. Available at: <https://doi.org/10.1016/0098-1354(95)00228-6>.

RUDD, K.; FERRARI, S. A constrained integration (CINT) approach to solving partial differential equations using artificial neural networks. **Neurocomputing**, Elsevier BV, v. 155, p. 277–285, May 2015. Available at: <https://doi.org/10.1016/j.neucom.2014.11.058>.

RUDY, S.; ALLA, A.; BRUNTON, S. L.; KUTZ, J. N. Data-driven identification of parametric partial differential equations. **SIAM Journal on Applied Dynamical Systems**, Society for Industrial & Applied Mathematics (SIAM), v. 18, n. 2, p. 643–660, Jan. 2019. Available at: <https://doi.org/10.1137/18m1191944>.

SEO, J. K.; KIM, K. C.; JARGAL, A.; LEE, K.; HARRACH, B. A Learning-Based Method for Solving Ill-Posed Nonlinear Inverse Problems: A Simulation Study of Lung EIT. **SIAM Journal on Imaging Sciences**, v. 12, n. 3, p. 1275–1295, Jan. 2019. ISSN 1936-4954. Available at: <https://epubs.siam.org/doi/10.1137/18M1222600>.

SMAOUI, N.; AL-ENEZI, S. Modelling the dynamics of nonlinear partial differential equations using neural networks. **Journal of Computational and Applied Mathematics**, Elsevier BV, v. 170, n. 1, p. 27–58, Sep. 2004. Available at: <https://doi.org/10.1016/j.cam.2003.12.045>.

TARANTOLA, A. **Inverse Problem Theory and Methods for Model Parameter Estimation**. Society for Industrial and Applied Mathematics, 2005. Available at: <https://doi.org/10.1137/1.9780898717921>.

TARTAKOVSKY, A. M.; BARAJAS-SOLANO, D. A.; HE, Q. Physics-Informed Machine Learning with Conditional Karhunen-Loève Expansions. **arXiv e-prints**, p. arXiv:1912.02248, Dec 2019.

TARTAKOVSKY, A. M.; MARRERO, C. O.; PERDIKARIS, P.; TARTAKOVSKY, G. D.; BARAJAS-SOLANO, D. Learning Parameters and Constitutive Relationships with Physics Informed Deep Neural Networks. **arXiv e-prints**, p. arXiv:1808.03398, Aug 2018.

TIPIREDDY, R.; PERDIKARIS, P.; STINIS, P.; TARTAKOVSKY, A. A comparative study of physics-informed neural network models for learning unknown dynamics and constitutive relations. **arXiv e-prints**, p. arXiv:1904.04058, Apr 2019.

TSOULOS, I. G.; GAVRILIS, D.; GLAVAS, E. Solving differential equations with constructed neural networks. **Neurocomputing**, Elsevier BV, v. 72, n. 10-12, p. 2385–2391, Jun. 2009. Available at: <https://doi.org/10.1016/j.neucom.2008.12.004>.

VARAH, J. M. A Spline Least Squares Method for Numerical Parameter Estimation in Differential Equations. **SIAM Journal on Scientific and Statistical Computing**, v. 3, n. 1, p. 28–46, Mar. 1982. ISSN 0196-5204, 2168-3417. Available at: <http://epubs.siam.org/doi/10.1137/0903003>.

VARZIRI, M. S.; MCAULEY, K. B.; MCLELLAN, P. J. Parameter and state estimation in nonlinear stochastic continuous-time dynamic models with unknown disturbance intensity. **The Canadian Journal of Chemical Engineering**, v. 86, n. 5, p. 828–837, Oct. 2008. ISSN 00084034. Available at: <http://doi.wiley.com/10.1002/cjce.20100>.

VINOD, A. V.; KUMAR, K. A.; REDDY, G. V. Simulation of biodegradation process in a fluidized bed bioreactor using genetic algorithm trained feedforward neural network. **Biochemical Engineering Journal**, Elsevier BV, v. 46, n. 1, p. 12–20, Sep. 2009. Available at: <https://doi.org/10.1016/j.bej.2009.04.006>.

VOGEL, C. R. **Computational Methods for Inverse Problems**. Society for Industrial and Applied Mathematics, 2002. Available at: <https://doi.org/10.1137/1.9780898717570>.

XUN, X.; CAO, J.; MALLICK, B.; MAITY, A.; CARROLL, R. J. Parameter Estimation of Partial Differential Equation Models. **Journal of the American Statistical Association**, v. 108, n. 503, p. 1009–1020, Sep. 2013. ISSN 0162-1459, 1537-274X. Available at: <http://www.tandfonline.com/doi/abs/10.1080/01621459.2013.794730>.

YADAV, N.; YADAV, A.; KIM, J. H. Numerical solution of unsteady advection dispersion equation arising in contaminant transport through porous media using neural networks. **Computers & Mathematics with Applications**, Elsevier BV, v. 72, n. 4, p. 1021–1030, Aug. 2016. Available at: <https://doi.org/10.1016/j.camwa.2016.06.014>.

ZHANG, X.; CAO, J.; CARROLL, R. J. Estimating varying coefficients for partial differential equation models: Varying Coefficients PDE Models. **Biometrics**, v. 73, n. 3, p. 949–959, Sep. 2017. ISSN 0006341X. Available at: <http://doi.wiley.com/10.1111/biom.12646>.