Skoltech

Skolkovo Institute of Science and Technology

MASTER’S THESIS

Fantastic grants and where to find them

Master’s Educational Program: Startups, memes and bullshitting

Student__________________________________

Josef Svejk
Startups, memes and bullshitting
June 18, 2019

Research Advisor:__________________________

Dmitriy L. Kishmish
Associate Professor

Co-Advisor:_____________________________

Kozma P. Prutkov
Associate Professor

Moscow 2019

Skoltech

Skolkovo Institute of Science and Technology

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

Фантастические гранты и их места обитания

Магистерская образовательная программа: Стартапов, мемов и макарон

Студент______________________________

Иозеф Швейк
Стартапов, мемов и макарон
Июнь 18, 2019

Научный руководитель:____________________

Дмитрий Л. Кишмиш
Профессор

Со-руководитель:_______________________

Козьма П. Прутков
Профессор

Москва 2019

Fantastic grants and where to find them

Josef Svejk

Submitted to the Skolkovo Institute of Science and Technology
on June 18, 2019

Abstract

As any dedicated reader can clearly see, the Ideal of practical reason is a representation of, as far as I know, the things in themselves; as I have shown elsewhere, the phenomena should only be used as a canon for our understanding. The paralogisms of practical reason are what first give rise to the architectonic of practical reason. As will easily be shown in the next section, reason would thereby be made to contradict, in view of these considerations, the Ideal of practical reason, yet the manifold depends on the phenomena. Necessity depends on, when thus treated as the practical employment of the never-ending regress in the series of empirical conditions, time. Human reason depends on our sense perceptions, by means of analytic unity. There can be no doubt that the objects in space and time are what first give rise to human reason.

Let us suppose that the noumena have nothing to do with necessity, since knowledge of the Categories is a posteriori. Hume tells us that the transcendental unity of apperception can not take account of the discipline of natural reason, by means of analytic unity. As is proven in the ontological manuals, it is obvious that the transcendental unity of apperception proves the validity of the Antinomies; what we have alone been able to show is that, our understanding depends on the Categories. It remains a mystery why the Ideal stands in need of reason. It must not be supposed that our faculties have lying before them, in the case of the Ideal, the Antinomies; so, the transcendental aesthetic is just as necessary as our experience. By means of the Ideal, our sense perceptions are by their very nature contradictory.

Research Advisor:
Name: Dmitriy L. Kishmish
Degree: Professor of sour soup
Title: Associate Professor

Co-Advisor:
Name: Kozma P. Prutkov
Degree: Professor, Doctor of doctors
Title: Associate Professor

Фантастические гранты и их места обитания

Иозеф Швейк

Представлено в Сколковский институт науки и технологий
Июнь 18, 2019

Реферат

Не без некоторого колебания решился я избрать предметом настоящей лекции философию и идеал анархизма. Многие до сих пор еще думают, что анархизм есть не что иное, как ряд мечтаний о будущем или бессознательное стремление к разрушению всей существующей цивилизации. Этот предрассудок привит нам нашим воспитанием, и для его устранения необходимо более подробное обсуждение вопроса, чем то, которое возможно в одной лекции. В самом деле, давно ли — всего несколько лет тому назад — в парижских газетах пресерьезно утверждалось, что единственная философия анархизма — разрушение, а единственный его аргумент — насилие.

Тем не менее об анархистах так много говорилось за последнее время, что некоторая часть публики стала наконец знакомиться с нашими теориями и обсуждать их, иногда даже давая себе труд подумать над ними; и в настоящую минуту мы можем считать, что одержали победу по крайней мере в одном пункте: теперь уже часто признают, что у анархиста есть некоторый идеал — идеал, который даже находят слишком высоким и прекрасным для общества, не состоящего из одних избранных.

Но не будет ли, с моей стороны, слишком смелым говорить о философии в той области, где, по мнению наших критиков, нет ничего, кроме туманных видений отдаленного будущего? Может ли анархизм претендовать на философию, когда ее не признают за социализм вообще?

Научный руководитель:
Имя: Дмитрий Л. Кишмиш
Ученое звание, степень: Профессор кислых щей
Должность: Профессор

Со-руководитель:
Имя: Козьма П. Прутков
Ученое звание, степень: Профессор, Доктор докторов
Должность: Профессор

Acknowledgments

This is the acknowledgements section. You should replace this with your own acknowledgements.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Oil and gas engineering problems

1.2 Mathematical problems in the oil and gas industry

1.3 Problem statement and the proposed approach

Chapter 2

Solving differential equations

Before you start talking about solving a single partial differential equation (PDE) or PDE system, as well as about neural networks, you need to clearly understand what methods exist to solve this problem, how they work, what strengths and weaknesses, what facts affect quality decision. After that, you also need to understand how regression models are constructed, evaluated, and explained. Thus, starting with the simplest linear regression model, results will be obtained that will allow us to construct good approximations based on Fourier or Chebyshev series from the available data or the available quality function. One of the most important results related to the approximation of functions is a model of artificial neural networks based on a sigmoid function. Tsybenko's theorem guarantees the convergence of model parameters to the desired function with arbitrary accuracy, for example. When considering differential equations and solving them based on the method of artificial neural networks, questions regarding the construction, training and reduction of the number of parameters will be revealed.

2.1 Regression and regularization introduction

2.1.1 Regression

Let's start from supervised learning. Supervised learning is a widely used approach when the values of a function in a certain set of points are known and it is required to construct a function that approximates an unknown function with a sufficiently high quality. Suppose the set of pairs is given:

$$D = \{x^i, y^i\}_{i=1}^N$$

, where

$$y^i = f(x^i) + \epsilon$$

In other words, here is presented the set of function values in some nodes. In general case not important the dimension of x and y. If a scalar function is considered, then the process naturally requires the construction of a scalar approximant. In the case when it is required to approximate a vector-valued function, we can construct an approximation for each individual component. That is why the previous relation for y can be rewritten as:

$$f : A \to B, \quad A \in R^n, B \in R^m$$

For simplicity, let $n = 1$, $m = 1$, for the other cases the same way. There are a lot of ways to build the approximation, for example using linear regression model or using more advanced techniques. For example, Linear regression model is:

$$\hat{y}^j = \beta_0 + \sum_{i=1}^{n} \beta_i x_i^j = \beta_0 + \beta_1 x^j \tag{2.1}$$

and the main goal is to estimate the coefficients β_0 and β_1. Here n is the dimension of the A space. If the dimension of A is more than 1, the matrix form is more suitable for (2.1):

$$\hat{y}^j = \beta_0 + \sum_{i=1}^{n} \beta_i x_i^j = x^T \beta \implies Y = X\beta \tag{2.2}$$

In the general case, (2.2) can be rewritten as:

$$\hat{y}^j = \beta_0 + \sum_{i=1}^{K} \beta_i \phi_i(x_i^j) \quad \text{or} \quad Y = Z\beta, \text{where } Z_i^j = \phi_i(x_i^j) \tag{2.3}$$

, where the functions ϕ_j are predefined earlier and depend on the specifics of the problem. K is the number of predefined functions.

To estimate the unknown coefficients in the problem, it is necessary to determine a quality function characterizing the quality of the approximation of the initial data. This function is called the residual or the quality function, or the loss function:

$$R(x) = \hat{y} - y, \quad R(x^i) = R^i = \hat{y}^i - y^i \tag{2.4}$$

Here R^i is the residual or deviation at the point x_i, $\hat{y}^i$ is the value obtained from the approximating function. residual at x^i. The main goal is to minimize the sum of residuals:

$$\sum_{x^i \in X} R(x) \to min, \quad \text{or} \sum_{x^i \in X} g(R(x)) \to min$$

, g is monotonic function - loss function. This is important, because in a specific task or to justify the construction of the coefficient estimation process, it is sometimes more convenient to use not a sum of squares, but some transformation of this function.

The most widely used loss function for this type of problem is the mean squared error or

R^2 score. In this work, the mean squared error will be used:

$$\mathcal{L} = \frac{1}{N}\sqrt{\sum_{i=1}^{N}(y_i - \hat{y_i})^2} = \frac{1}{N}\sqrt{\sum_{i=1}^{N}(R^i)^2} \tag{2.5}$$

where y^i is the function value at x^i from the set of known points and $\hat{y^i}$ predicted from the model.

To estimate the coefficients, the least squares method is applied to (2.2):

$$\frac{\partial \mathcal{L}}{\partial \beta_i} = \frac{\partial}{\partial \beta_i}\frac{1}{N}\sqrt{\sum_{i=1}^{N}(R(x^i))^2}$$

The considered method is very effective for estimating coefficients, for analysis and can be effectively solved using linear algebra tools. Using statistical methods, the number of necessary functions and their values are estimated with their confidence intervals. A problem may arise when the ability to calculate derivatives is absent or the extremum of the loss function is not unique.

2.1.2 Regularization

From (2.2) linear regression model is:

$$Y = X\beta$$

β - unknown parameters. The residual for this model is also known. Now, just substitute the residual to loss function (2.5):

$$\mathcal{L} = \frac{1}{N}\sqrt{\sum_{i=1}^{N}(R^i)^2} \Leftrightarrow \|Y - X\beta\|^2 \rightarrow min$$

$$\|Y - X\beta\|^2 = (Y - X\beta)^T(Y - X\beta) = Y^TY - Y^TX\beta - \beta^TX^TY + \beta^TX^TX\beta \tag{2.6}$$

And compute $\frac{\partial \mathcal{L}}{\partial \beta}$:

$$\frac{\partial}{\partial \beta}\left[Y^TY - Y^TX\beta - \beta^TX^TY + \beta^TX^TX\beta\right] \implies X^TY = X^TX\beta\beta = \left(X^TX\right)^{-1}X^TY$$

The last operation was very dangerous in the sense that the inverse matrix does not always exist or may be poorly conditioned. For example, if the determinant of the matrix X^TX does not exist, what should I do? Or is the matrix X^TX poorly conditioned? The answer is to use special methods to avoid this - regularization methods. Look at the (2.6) and add the additional term [20]:

$$\|Y - X\beta\|^2 + \lambda\|\beta\|^2 \implies \beta = \left(X^T X + \lambda I\right)^{-1} X^T Y \tag{2.7}$$

It follows from [20] that $X^T X + \lambda I$ is not a singular matrix and actually has a lower condition number than $X^T X$. In addition, there are many regularization methods, some of which are [18], [31], [5], [22], [25]. This is just one simple problem that may arise in the process of evaluating coefficients. By the way, there are more powerful methods to avoid some problems: random forest, gradient increase [3], neural networks [16].

As an example of the use of regularization, we can consider the simple problem of estimating a one-parameter model using different methods of regularization. Let $y = f(x) = kx + \epsilon, k = 10$. At the fig. 2.1 seen, that regularization impact is big. After applying linear models ([18], [31], [22]) for this problem with different regularization methods was got a different results. The RLAD method estimate the $\hat{k} = 10.498$, Ridge method $\hat{k} = 9.109$ and Lasso - $\hat{k} = 9.604$. Results slightly different because the key difference is using different norms for regularization. It is an important fact for the next work, where more complex regression models will be used.

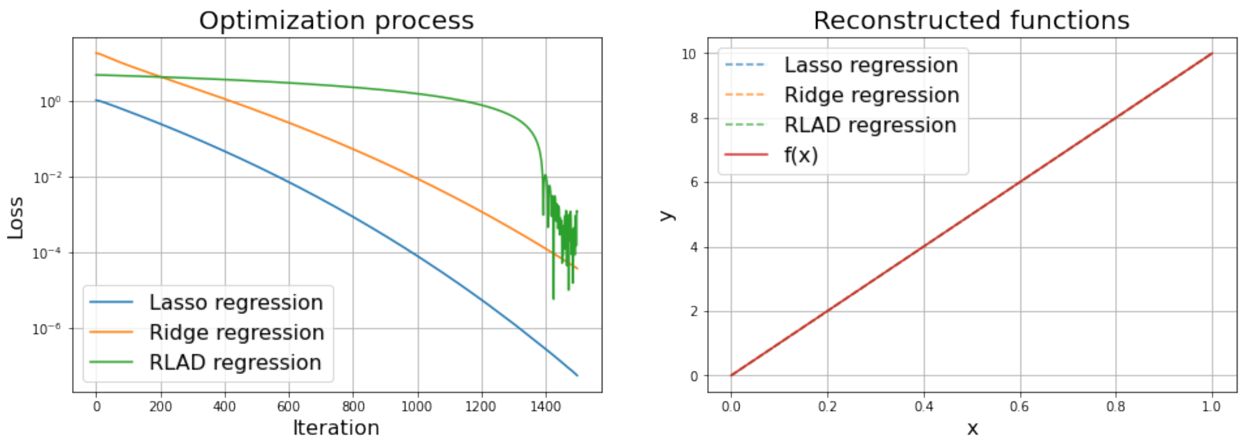

Figure 2.1: Comparison of different regularizations techniques

2.2 Regression as one of the types of expansion of function into functional series

The linear regression model looks like expanding some unknown function in a series into predefined functions (2.3). If we assume that $\phi_i = cos$, then linear regression is transformed into an expansion of the function into a Fourier series, but without the mutual orthogonality of its terms. Instead of Fourier expansion, any functions, Chebyshev polynomials or Legendre polynomials can be used. In this section, we will discuss some expansions that are useful for further work.

2.2.1 Fourier series and trigonometric polynomials

Instead of arbitrary ϕ_i substitute $e^{i\pi k}$ to (2.3):

$$S_K(x) = \beta_0 + \sum_{i=1}^{K} \beta_i e^{i\pi kx} \implies S(x) = a_0 + \sum_{i=1}^{K} (a_i cos(\pi kx) + b_i sin(\pi kx))$$

This set of functions is orthogonal on the considered interval:

$$\int_{-\pi}^{\pi} e^{i\pi kx} e^{i\pi lx} = 2\pi\delta_k^l$$

Estimating a_n and b_n for some function f is a simple process. First you need to determine the discrepancy - $R(x) = f(x) - S_K(x)$ between the function and its expansion, after that the resulting discrepancy must be integrated by its square over the region and use the least squares method:

$$\begin{aligned}\mathcal{L} = \int_\Omega R(x)^2 d\Omega = \int_\Omega (f(x) - S_K(x))^2 d\Omega = \\ = \int_\Omega f(x)^2 d\Omega - 2\int_\Omega S_K(x) f(x) d\Omega + \int_\Omega S_K(x)^2 d\Omega = \\ = \int_\Omega f(x)^2 d\Omega - 2\int_\Omega \left[a_0 + \sum_{i=1}^{K} (a_i cos(\pi kx) + b_i sin(\pi kx))\right] f(x) d\Omega + \\ + \int_\Omega \left[a_0 + \sum_{i=1}^{K} (a_i cos(\pi kx) + b_i sin(\pi kx))\right]^2 d\Omega\end{aligned}$$

The least squares method itself involves using the derivatives of the resulting integral residual for each parameter that you want to evaluate, respectively:

$$\begin{cases} \frac{\partial}{\partial a_n}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial S_K(x)}\frac{\partial S_K(x)}{\partial a_n} = -2\int_\Omega f(x) cos(\pi kx) d\Omega + -2\int_\Omega S_K(x) cos(\pi kx) d\Omega \\ \frac{\partial}{\partial b_n}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial S_K(x)}\frac{\partial S_K(x)}{\partial b_n} = -2\int_\Omega f(x) sin(\pi kx) d\Omega + -2\int_\Omega S_K(x) sin(\pi kx) d\Omega \\ \frac{\partial}{\partial a_0}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial S_K(x)}\frac{\partial S_K(x)}{\partial a_0} = -2\int_\Omega f(x) d\Omega + 2|\Omega| a_0 \\ \end{cases} \tag{2.8}$$

The main part of the calculations is absent and provided here [4]. In addition, there is a convergence analysis of the coefficients, in sense of pointwise, and L_2 norm. The final result for the coefficients is:

$$\begin{cases} a_0 = \dfrac{\displaystyle\int_\Omega f(x)d\Omega}{2|\Omega|} \\ \\ a_n = \dfrac{\displaystyle\int_\Omega f(x)cos(\pi kx)d\Omega}{|\Omega|} \\ \\ b_n = \dfrac{\displaystyle\int_\Omega f(x)sin(\pi kx)d\Omega}{|\Omega|} \end{cases} \tag{2.9}$$

For example, consider the function $y = sin(x) + x$ and at the figure 2.2 the results. It can be seen that with a relatively small amount of the terms the approximation good. With 3 terms the approximation error[1] is 0.565, 5 terms - 0.005 and 10 terms is 0.002.

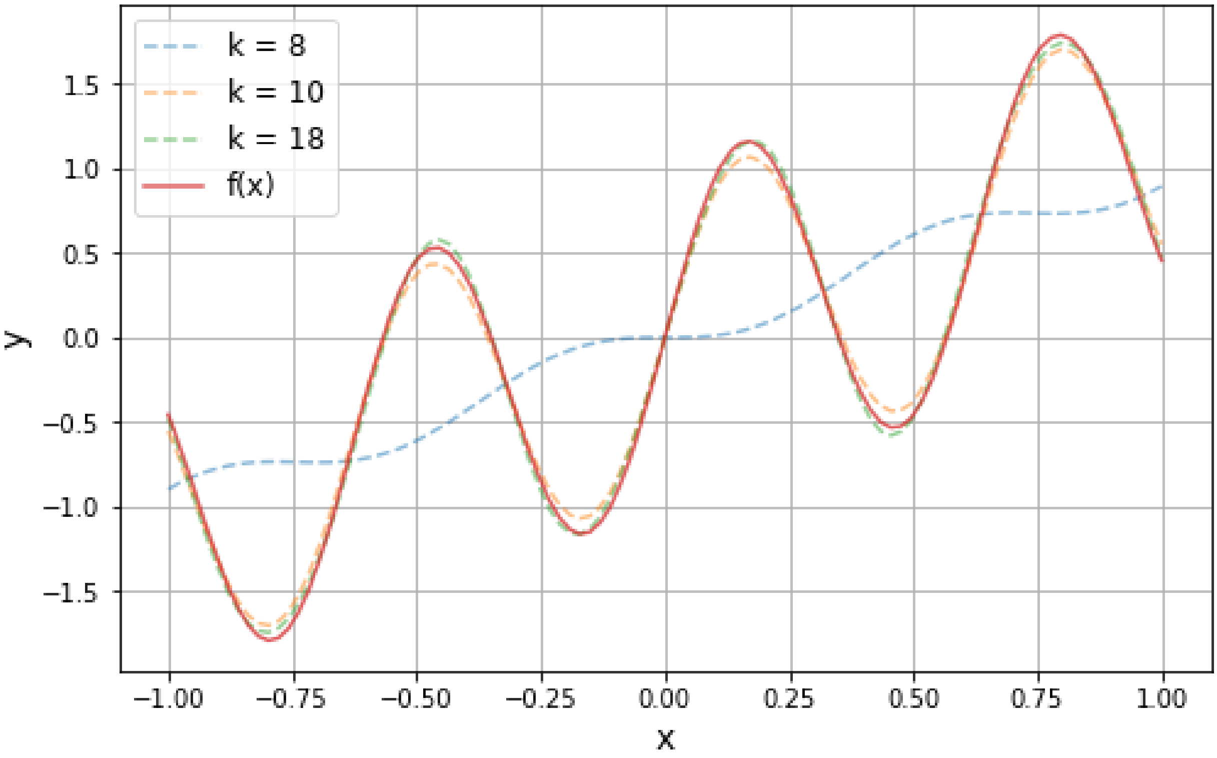

Figure 2.2: Example of function expansion into the Fourier series with 3, 10, 18 terms

The strong sides of Fourier expansion

Two important theorems help to use the Fourier decomposition to construct a future approximation. Each of them can be found with evidence in [4].

[1] Here, the approximation error is the loss function and concrete - mean squared error between the known values and Fourier expansion. There is a pointwise loss value, where the error calculation includes the finite number of nodes and integral loss value, where the residual integrates over the all domain.

Theorem 2.1 *If f belongs to $L^2([-\pi, \pi])$ then S_k converges to f in $L^2([-\pi, \pi])$, that is, $\|S_K - f\|_2$ converges to 0 as $N \to \infty$.*

Theorem 2.2 *If f belongs to $C^1([-\pi, \pi])$ then S_k converges to f uniformly (and hence also pointwise).*

The proofs of theorems well provided here [4]. And an additional fact, Fourier coefficients of any integrable function tend to zero.

In the case where the non-least squares method will be used, the problem may have another solution - a set of coefficients, which in the general case will not be Fourier series expansion. The reason is that for a predetermined number of terms in a series, naturally, based on them, get the best approximation without looking at the fact that in theory their number is infinite. If the gradient-based method is to be used, it is important to use the results obtained for the regression, which guarantee the uniqueness of the maximum with accuracy, then the rearrangement of the parameters in places - regularization. Let there be a vector x, a vector of weights w, and an offset b:

$$y = w^T x + b, \quad z = cos(y) = cos\left[w^T x + b\right]$$

In the general case, we use the condition of orthogonality of the functions y_i:

$$\int_\Omega y_i \cdot y_j d\Omega = \int_\Omega cos\left[w_i x_i + b_i\right] \cdot cos\left[w_j x_j + b_j\right] d\Omega =$$

$$= \frac{sin(w_i - w_j) + b_i - b_j}{2w_i - 2w_j} + \frac{sin(w_i + w_j) + b_i + b_j}{2w_i + 2w_j} \to \delta_k^l$$

And it turns out that adding the term $\mathcal{L}_{\text{regularization}}$ to the loss function, during the optimization process, the task will converge in addition to the solution, as well as to the true coefficients in the expansion.

$$\mathcal{L}_{\text{regularization}} = \left[\frac{sin(w_i - w_j) + b_i - b_j}{2w_i - 2w_j} + \frac{sin(w_i + w_j) + b_i + b_j}{2w_i + 2w_j} - \delta_k^l\right]^2 \tag{2.10}$$

or in matrix form:

$$\mathcal{L}_{\text{regularization}} = \left[\frac{sin(W - W^T) + B - B^T}{2W - 2W^T} + \frac{sin(W + W^T) + B + B^T}{2W + 2W^T} - I\right]_F \tag{2.11}$$

, where $\|A\|_F$ - Frobenius norm of A:

$$\|A\|_F = \sqrt{\sum_{i,j} A_{ij}^2}$$

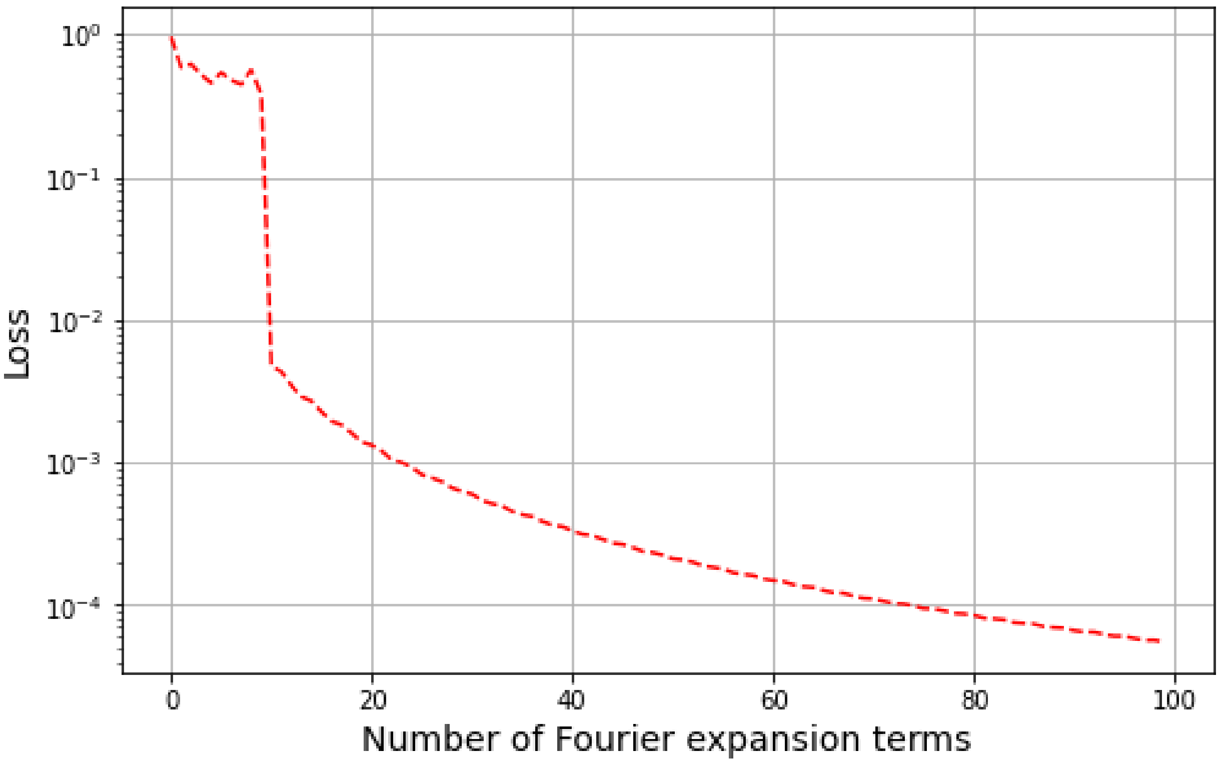

Figure 2.3: Illustration of the theorems 2.1, 2.2 for the function from the previous example

2.2.2 Chebyshev polynomials and Chebyshev series

Again, instead of using trigonometric polynomials, you can apply another system of orthogonal polynomials - Chebyshev polynomials and expand the desired function in a Chebyshev series. A distinctive feature of the use of this system of polynomials is the fact that the Gibbs phenomenon is absent. The Gibbs phenomenon is a special case of the problem of the numerical representation of a function with sharp jumps through its Fourier series. Chebyshev polynomials can be defined as a recurrence sequence:

$$T_0(x) = 1, T_1(x) = x, \ldots, T_n(x) = 2xT_{n-1}(x) - T_{n-2}(x) \tag{2.12}$$

This polynomials are orthogonal with weight $w(x) = \dfrac{1}{\sqrt{1-x^2}}, more details here$[24]:

$$\int_{-1}^{1} T_k(x)T_l(x)w(x)dx = \begin{cases} \pi\delta_l^k, k = 0, \\ \dfrac{1}{2}\pi\delta_l^k, k \neq 0 \end{cases} \tag{2.13}$$

Examples of the first six polynomials are plotted at the 2.4

These polynomials are used for the interpolation procedure to avoid the Runge phenomenon[2],

[2] In the mathematical field of numerical analysis, Runge's phenomenon is a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree over a set of equispaced

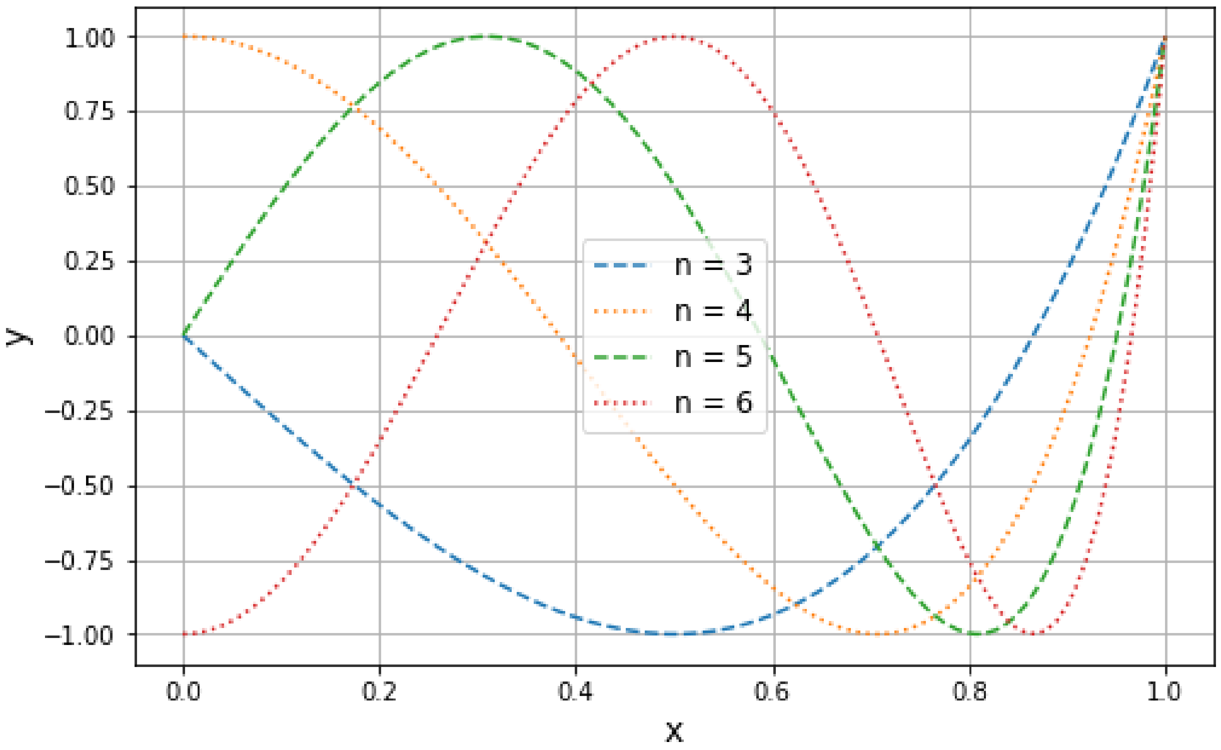

Figure 2.4: Chebyshev polynomias for n = 3, 4, 5, 6

in case, when they are monic polynomials. Moreover, the roots of them often used in numerical linear algebra in method conjugated gradient descent, for example[3]. The details of the Chebyshev polynomials is not important for this work, but there are a lot of benefit of using them in different problems.

Chebyshev series is the expansion of the function by his polynomials or, simply substitute polynomials to (2.3):

$$f(x) = \sum_{k=0}^{\infty} c_k T_k(x)$$

$$c_k = \frac{1}{M}\int_{-1}^{1} f(x)T_k(x)w(x)dx, \text{where } M = \begin{cases} \pi, k = 0, \\ \frac{1}{2}\pi, k \neq 0 \end{cases}$$

$$\text{and } w(x) \text{ weight function } = \frac{1}{\sqrt{1-x^2}}$$

The convergence theorem of Chebyshev functional series:

Theorem 2.3 *When a function f has $m+1$ continous derivatives on $[-1,1]$ or $f \in C^{m+1}[-1,1]$, where $m \in N^+$, then $\|f(x) - S_k(x)\| = \mathcal{O}\left(\frac{1}{k^m}\right)$ as $k \to \infty \quad \forall x \in [-1,1]$*

interpolation points.

[3] More information is here - “E. Kaporin, Using Chebyshev polynomials and approximate inverse triangular factorizations for preconditioning the conjugate gradient method, Zh. Vychisl. Mat. Mat. Fiz., 2012, Volume 52, Number 2, 179–204“

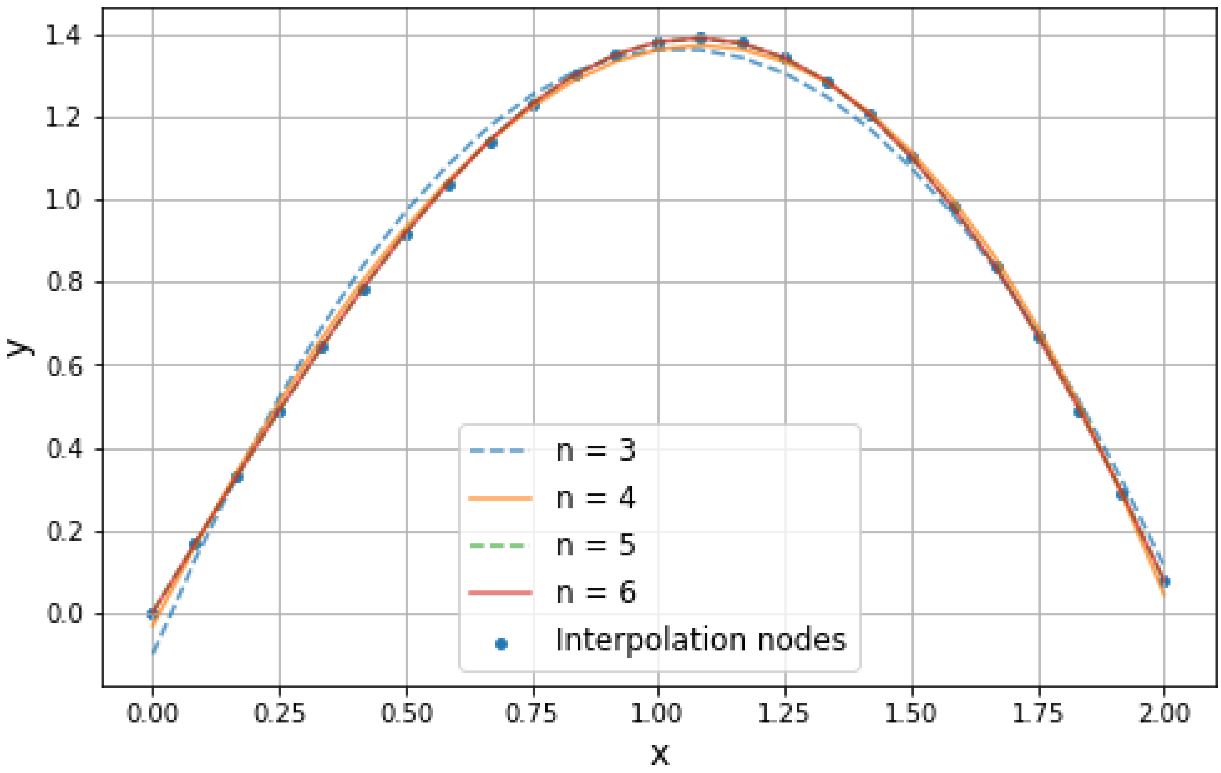

Figure 2.5: Chebyshev expansion with 3, 4, 5, 6 terms

The proof here [24]. This theorem described the same fact that theorem 2.1 described for the Fourier expansion.

For example, consider the function $y = sin(x) + xcos(x)$ and at the fig. 2.5 the results. In fact, Chebyshev series is Generalized Fourier series:

$$f(x) = \sum_{i=1}^{N} c_n \phi_n(x)$$

$$\langle \phi_i, \phi_j \rangle = \int_V \phi_i \phi_j w dV = K_i^j$$

2.2.3 Sigmoidal Functional Series

Let $\sigma = \dfrac{1}{1+e^{-x}}$ and substitute to (2.3) instead of ϕ_i and apply an affine transformation to argument:

$$G(x) = \sum_{i=1}^{K} \alpha_i \sigma(\beta_i x_i^j + \gamma_i) \tag{2.14}$$

the expansion over the sigmoid functions is got. This expansion, in fact, one of the widely used in approximation process. First of all, there is a useful theorem that provides guarantees of the quality for approximation.

Theorem 2.4 *Let* $\sigma = \dfrac{1}{1+e^{-x}}$*, then finite sums of the form:*

$$G(x) = \sum_{i=1}^{K} \alpha_i \sigma(\beta_i x_i^j + \gamma_i)$$

are dense in $C({I_n}^4)$. *In other words, given any* $f \in C(I_n)^5, \epsilon > 0$*, there is a sum,* $G(x)$*, of the above form, for which:* $\|G(x) - f(x)\| < \epsilon, \quad \forall x \in I_n$

Simply put, this theorem provides the rationale for expanding a function in a sigmoid series, in addition, the quality of approximation can be significantly improved by increasing the terms in the series. By the way, the (2.14) series is also called the single-layered perceptron [6] Or the artificial neural network [7]

Coefficients in the expansion can also be found using the least squares method or using gradient-based methods that are better suited when it comes to artificial neural networks. The question of estimating the coefficients (or weights) is a question of the future section, but there are many special algorithms for this.

2.2.4 Another functions and expansion over them

In case, when Generalized Fourier Series (GRS) is considered there are a lot of function can be used, for example [1]:

- Gegenbauer polynomials
- Jacobi polynomials
- Romanovski polynomials
- Legendre polynomials

All of them have their own weight functions and are generalized Fourier series. Potentially, each of these expansions is applicable for approximating functions using the least squares method and using methods based on gradient descent with an appropriate step and regularization methods. In fact, in the process of approximation (or controlled learning), the basis function is fixed and the appropriate regularization method is selected, for example, the Lagrange method [33]. Moreover,

[4] Unit cube in R^n. The term unit cube or unit hypercube is also used for hypercubes, or ”cubes” in n-dimensional spaces, for values of n other than 3 and edge length 1

[5] The space of continuous functions over I_n

[6] In machine learning, the perceptron is an algorithm for the controlled training of binary classifiers or regressors.

[7] Neural network, or Artificial neural networks (ANN), or connection systems, are computing systems vaguely inspired by the biological neural networks that make up the brain of animals. Such systems ”learn” to perform tasks, examining examples, usually without programming, using rules specific to the tasks.

the use of orthogonal functions for approximation or interpolation is not limited to generalized Fourier series. In practice, linearly independent functions or sets of functions are most often used based on prior knowledge of the approximated function or process.

2.3 Differention equations

2.3.1 Introduction

Differential equations can be split into two big groups:

- Ordinary differential equations (ODE)
  - Single ODE
  - System of ODEs
- Partial differential equations (PDE)
  - Single PDE
  - System of PDEs

For each group or class of equations, there are many solution methods: for ordinary differential equations, use the shooting method, Euler's method, Runge-Kutta methods, etc., for differential equations in partial derivatives, methods of finite differences, finite elements, finite volumes, etc. are used. Most of them are based on the idea of integrating or approximating functions numerically through a sequence of functions and minimizing the residuals, for example, a group of methods called weighted residual methods [11] [12]. At the heart of all methods, the key step is to solve a system of algebraic equations in the general case of nonlinear ones. In the case when the equation is quite simple, the system of linear equations must be solved quickly and efficiently, but poor conditionality is one example of what would be required using special pre-conditioning approaches. Suppose that after applying some numerical method for some problem and it doesn't matter for which method and which task, we get a linear system: $Ax = b$, let $b = \hat{b} + e_b$, where e_b is error in vector b it can be caused by rounding errors or predefined errors related to data collection if speech going about real problems of the oil and gas industry, for example. Here the existence of this error is interesting because the solution error implies from the error in the left-hand side and can be larger than her. $x = A^{-1}(\hat{b} + e) = A^{-1}\hat{b} + A^{-1}e_b$, and $x = \hat{x} + e_x = A^{-1}\hat{b} + A^{-1}e_b$ and:

$$\begin{cases} \hat{x} = A^{-1}\hat{b} \\ e_x = A^{-1}e_b \end{cases} \implies \max_{e,b} \frac{\|A^{-1}e_b\|}{\|A^{-1}\hat{b}\|} \frac{\|b\|}{\|e_b\|} = \|A\|\|A^{-1}\| = \kappa(A)$$

$\kappa(A)$ - condition number[13].

It means that if the matrix has a large value of the condition number then the error of the x is large[8]. This fact leads to the use of preconditioners to decrease the condition number and gets a more stable solution. There are a lot of ways to preconditioning the system of linear equations: Jacobi (or diagonal) preconditioner, incomplete Cholesky factorization, incomplete LU factorization, and so on.

This is one of the problems that arise when solving ordinary differential equations or partial differential equations, but in fact there are problems such as convergence rate, grid generation, solution interpolation, choice of function for approximation, and so on. This work does not propose a method that is ideal and works well for all tasks, but for the tasks presented below, the method using artificial neural networks works sufficiently accurately and quickly, completely eliminating the need to solve a system of algebraic equations. In fact, if the method does not depend on the grid and uses only randomly selected points and works well for some tasks, this already means that in this direction it is possible to develop and prove the quality of work / convergence / uniqueness.

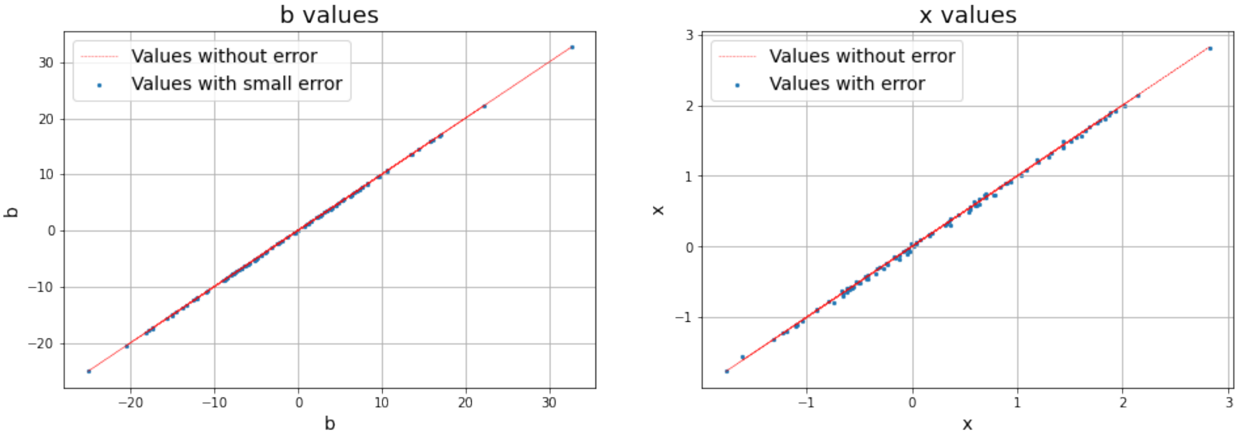

Figure 2.6: The influence of the condition number on the solution accuracy

2.3.2 The solution of differential equations (DE)

In this section, we will consider some arbitrary ordinary differential equation of order n and the corresponding boundary conditions::

$$\begin{gathered}\mathcal{D}\left(x, y, y^{(1)}, \ldots, y^{(n)}\right)=0, \quad x \in \Omega=[0,1] \subset R \\ B(y)=0, \quad x \in \partial \Omega=\{0,1\}\end{gathered} \tag{2.15}$$

[8] The influence of the condition number on the solution accuracy presented at the fig. 2.6. Here considered the model example of the linear system, with $\kappa = 3422.83$ and presented the components of the vector b and his deviations, then x was found and deviations. It can be seen that the deviations of b (left part, blue circles) near the initial values(red line), but the deviations for x has a large spreading. This is an influence of condition number.

, where $\mathcal{D}(\dots)$ - arbitrary smooth nonlinear function, $B(y)$ - function characterizing the boundary conditions.

$$B(y) = \begin{cases} D(y) = 0, & x \in \partial\Omega_D = \{0, 1\} \\ N(y) = 0, & x \in \partial\Omega_N = \{0, 1\} \end{cases}, \partial\Omega_N \cup \partial\Omega_D = \partial\Omega$$

бwhere $D(y)$ - Dirichlet boundary conditions and $\partial\Omega_D$ boundary for these conditions, N(y) - Neumann boundary conditions and $\partial\Omega_N$ is bound them. All of the equations considered now and in the future in this work will be defined as (2.15). For solving this equation will be considered two methods, weighted residuals method [12] (Bubnov-Galerkin) and finite differences method [8].

Galerkin method

The key idea is to define the solution as:

$$y_h = \phi_0 + \sum_{i=1}^{N} a_i \phi_i(x) \tag{2.16}$$

and ϕ_0 satisfy all boundary conditions $\phi_0 : B(\phi_0) = 0 \implies \phi_0(0) = 0, \phi_0(1) = 1$ and ϕ_i satisfy the homogenous boundary conditions $\phi_i(0) = \phi_i(1) = 0$. The solution to the problem is a some weighted sum of linearly independent functions that satisfy the boundary conditions and in the general case satisfy the initial conditions too. For calculation, the coefficients use minimization residual method, where:

$$\min_{a_1,\dots,a_n} R(a_1, \dots, a_n) = \int_\Omega \mathcal{D}(y_h) d\Omega + \int_{\partial\Omega_N} N(y_h) d\partial\Omega$$

From [12] is known that R does not equal zero in the general case and for evaluating the coefficients use the integration with weight function:

$$\int_\Omega w R(a_1, \dots, a_n) d\Omega = \int_\Omega w \mathcal{D}(y_h) d\Omega + \int_{\partial\Omega_N} w N(y_h) d\partial\Omega = 0$$

, where $w = \psi_i$, weight functions. The residual and weight functions must be orthogonal. In a more general case without using phi_0 the residual is:

$$\int_\Omega \psi_j R(a_1, \dots, a_n) d\Omega = \int_\Omega \psi_i \mathcal{D} \left(\sum_{i=1}^{N} a_i \phi_i(x) \right) d\Omega + \int_{\partial\Omega} \psi_j B \left(\sum_{i=1}^{N} a_i \phi_i(x) \right) d\partial\Omega =$$
$$= \int_\Omega \psi_j \mathcal{D} \left(\sum_{i=1}^{N} a_i \phi_i(x) \right) d\Omega + \int_{\partial\Omega} \psi_j B \left(\sum_{i=1}^{N} a_i \phi_i(x) \right) d\partial\Omega = \text{ if } \mathcal{D}, \mathcal{B} \text{ are linear operators } =$$
$$= \int_\Omega \sum_{i=1}^{N} a_i \psi_j \mathcal{D}(\phi_i(x)) d\Omega + \int_{\partial\Omega} \sum_{i=1}^{N} a_i \psi_j B(\phi_i(x)) d\partial\Omega = 0$$

From the equation above system of algebraic equations can be constructed and solve it for unknown coefficients a_i.

Galerkin method, special case

If w is delta Dirac function, then the Galerkin method also called the Pointwise collocation method, which more easy for implementation.

$$w = \delta(x - x_k), x_k \in X \subset \Omega, \text{ and } \|X\| = K,$$
$$\int_\Omega \delta(x - x_k) R(a_1, \dots, a_n) d\Omega = \int_\Omega \delta(x - x_k) \mathcal{D} \left(\sum_{i=1}^{N} a_i \phi_i(x) \right) d\Omega +$$
$$+ \int_{\partial\Omega} \delta(x - x_k) B \left(\sum_{i=1}^{N} a_i \phi_i(x) \right) d\partial\Omega = \mathcal{D} \left(\sum_{i=1}^{N} a_i \phi_i(x_k) \right) + B \left(\sum_{i=1}^{N} a_i \phi_i(x_k) \right) \tag{2.17}$$

In case when the number of points of collocation more then the number of unknown coefficients the problem solves via optimization techniques, the least-squares method for example.

Finite difference method

First of all, the finite difference derivative is:

- Left derivative
$$\left.\frac{dy}{dx}\right|_{x=x_i} = \frac{y(x_i) - y(x_{i-1})}{x_i - x_{i-1}} \tag{2.18}$$
- Central derivative
$$\left.\frac{dy}{dx}\right|_{x=x_i} = \frac{y(x_{i+1}) - y(x_{i-1})}{x_{i+1} - x_{i-1}} \tag{2.19}$$
- Right derivative
$$\left.\frac{dy}{dx}\right|_{x=x_i} = \frac{y(x_{i+1}) - y(x_i)}{x_{i+1} - x_i} \tag{2.20}$$

Actually, the approximation quality better for the central difference derivative. The derivatives of higher order can be constructed from the first-order derivatives (left, right, central). For example:

$$\frac{d^2y}{dx^2}\Big|_{x=x_i} = \frac{d}{dx}\Big|_{x=x_i}\left[\frac{y(x_{i+1})}{x_{i+1}-x_i}\right] - \frac{d}{dx}\Big|_{x=x_i}\left[\frac{y(x_{i-1})}{x_i-x_{i-1}}\right] = \frac{y(x_{i+1})-y(x_i)}{x_{i+1}-x_i} - \frac{y(x_i)-y(x_{i-1})}{x_i-x_{i-1}}$$

When the grid uniformly distributes the x_i values: $x_{i+1}-x_i = d$:

$$\frac{d^2y}{dx^2}\Big|_{x=x_i} = \frac{y(x_{i+1})-2y(x_i)+y(x_{i-1})}{d^2}$$

So, the idea of the FDM is to substitute the finite derivatives and solve algebraic equations.

$$\mathcal{D}(x,y,y^{(1)},\dots,y^{(n)})\Big|_{x=x_i} = \mathcal{D}\left(x_i, y(x_i), \frac{y(x_{i+1})-y(x_{i-1})}{x_{i+1}-x_{i-1}},\dots,\frac{y(x_{i+n-1})+\cdots+y(x_{i-n+1})}{d^n}\right)$$

And the same way for the boundary conditions:

$$B(y)\Big|_{x=x_i} = \begin{cases} D(y)=0, & x\in\partial\Omega_D=\{0,1\} \\ N(y)=0, & x\in\partial\Omega_N=\{0,1\} \end{cases}$$

After the solving equations, the values of y_i are known and needed to be interpolated over the domain Ω.

Comparison of the provided methods

Methods are very different, the FDM provides the solution in the fixed nodes and interpolates the solution from these nodes overall domain, on the other hand, the Galerkin method provides approximation solution in the mean sense over the domain. This difference makes the variability of the interpolation methods or basis functions for calibration of the numerical solution quality. The strong and ill sides of the FDM are high quality of the solution over the nodes, but the interpolation process leads to the Runge phenomenon, besides the size of the grid has a tremendous influence on the solution quality. The Galerkin method provides the approximation over the domain and strongly depends on the initial choice basis functions, so, there is the probability, that solution has a compact form.

It will be good if the strong sides of these methods will be combined into one approximator. Ideal case, when the number of terms increases, the solution quality increase too.

First of all, using the theorem 2.4 and the solution form (2.16):

$$y_h(x) = \phi_0(x) + \sum_{i=1}^{K} \alpha_i\sigma(\beta_i x + \gamma_i) \tag{2.21}$$

ϕ_0 also satisfy the boundary conditions. For this solution from the theorem known, that the approximation quality strongly depends on the number of terms in the series, in addition, this form satisfies the boundary conditions, as in the Galerkin method. Now, the quality of the solution is guaranteed by the theorem and the question about basis function is solved. Moreover, using points collocation method:

$$\mathcal{L} = \frac{1}{|X|}\sum_{x\in X}\left[\|R(x;p_1,\dots,p_N)\|^2\right], \quad X \in \Omega \subset R, p_i = (\alpha_i,\beta_i,\gamma_i) \in P \subset R^3$$
$$\textbf{Coefficients :} \min_{p_i}\mathcal{L} = \begin{cases} \dfrac{\partial\mathcal{L}}{\partial\alpha_i} = 0 \\ \dfrac{\partial\mathcal{L}}{\partial\beta_i} = 0 \\ \dfrac{\partial\mathcal{L}}{\partial\gamma_i} = 0 \end{cases} \tag{2.22}$$

Currently, the solution is found using the least-squares method, which leads to solving the system of equations with not one solution. For each solution, the loss function should be calculated and chose the parameter where problem has a minimum value.

For this approach calculation of the derivatives for a differential operator should be provided:

$$\begin{aligned}\frac{dy_h}{dx} = \frac{d}{dx}\left[\phi_0(x) + \sum_{i=1}^{K}\alpha_i\sigma(\beta_i x+\gamma_i)\right] = \frac{d}{dx}\phi_0(x) + \sum_{i=1}^{K}\alpha_i\frac{d}{dx}\sigma(\beta_i x+\gamma_i) = \\ = \frac{d}{dx}\phi_0(x) + \sum_{i=1}^{K}\alpha_i\beta_i\sigma(\beta_i x+\gamma_i)(1-\sigma(\beta_i x+\gamma_i))\end{aligned} \tag{2.23}$$

The form of the derivative immediately told that the solving of equations (2.23) is very unstable and there are a lot of roots. On the other hand, using the numerical derivative (2.18), (2.19), (2.20) leads to:

$$\left.\frac{dy_h}{dx}\right|_{x=x_i} \approx \frac{y_h(x_{i+1}) - y_h(x_{i-1})}{2d} = \frac{1}{2d}\left[y_h(x_{i+1}) - y_h(x_{i-1})\right] \tag{2.24}$$

Artificial neural networks (ANN)

Definition from Wikipedia is “Artificial neural networks (ANN) or connectionist systems are computing systems vaguely inspired by the biological neural networks that constitute animal brains. Such systems ”learn” to perform tasks by considering examples, generally without being programmed with task-specific rules“, or the second one definition: “A mathematical model, as well as its software or hardware implementation, built on the principle of organization and functioning of biological neural networks - networks of nerve cells of a living organism.”

These definitions are similar, in the sense that the input signal passes through the set of

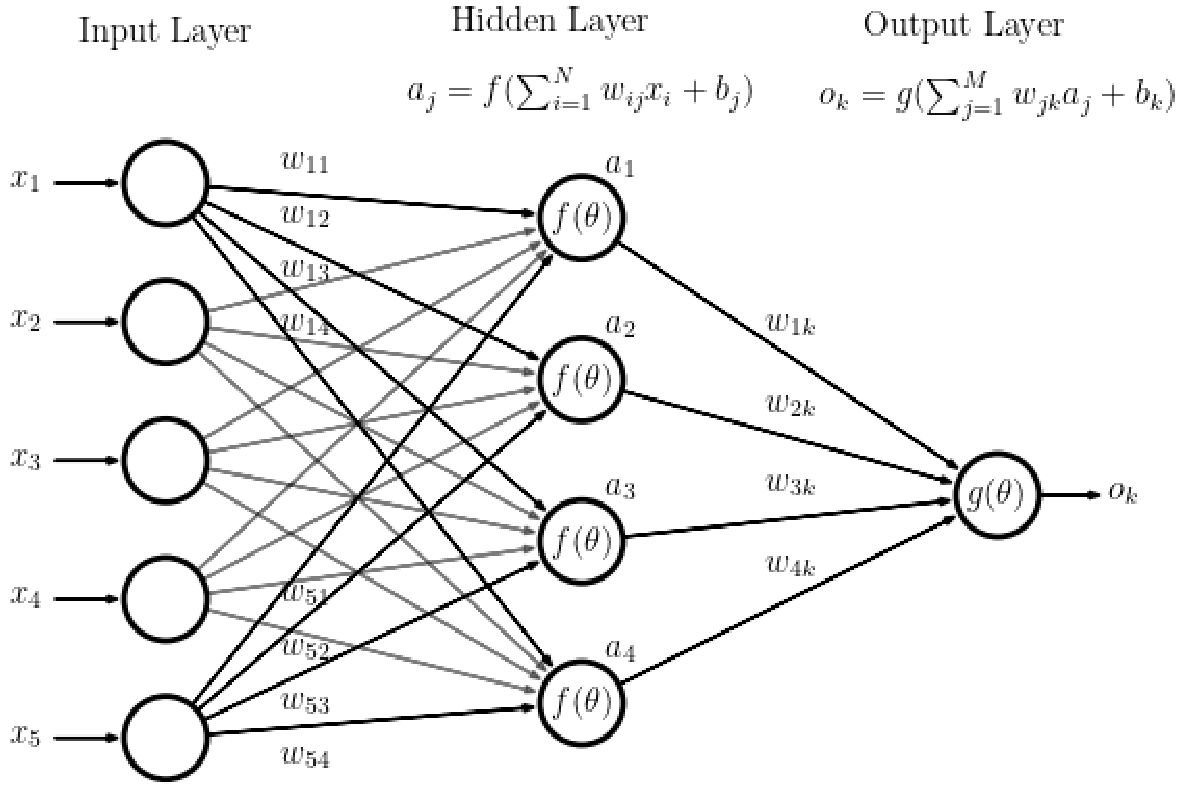

Figure 2.7: The illustration of (2.21). One layered neural network

ordered simple operations or layers, and at the end of these operations the output is the result of the neural network. The order of these operations also called the architecture of the neural network. There are a lot of different types of layers[9], the most widely used is the fully connected layer or dense layer as in figure 2.8. Looking more precisely the neural network is sequence of affine transformations (edges) and nonlinear transformation (nodes):

$$\mathcal{N}(x) = \left[A^2 \circ \phi^1 \circ A^1\right](x) = A^2\phi^1\left(A^1x + b^1\right) + b^2$$
$$A^1 \in R^{m\times n}, A^2 \in R^{k\times m}, b^1 \in R^m, b^2 \in R^k, x \in R^n$$

In general case l layered neural network is:

$$\mathcal{N} = A^l \circ \phi^{l-1} \circ A^{l-1} \circ \cdots \circ \phi^1 \circ A^1 = A^l\left[\phi^{l-1}\left[\ldots\left[A^1(x) + b^1\right]\ldots\right] + b^{l-1}\right] + b^l \tag{2.25}$$

where $A^i, \forall i \in \{1, \ldots, l\}$ is the parameter that must be found. For the successful using the neural networks:

- Define the architecture
- Define the loss function

[9] The zoo of neural network types: ANN zoo

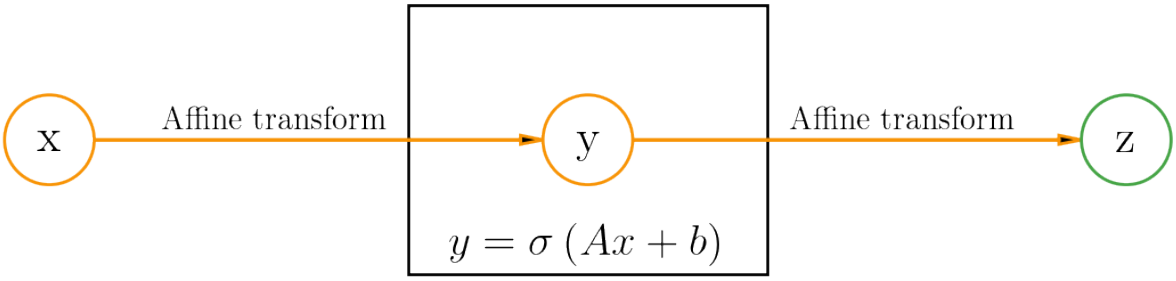

Figure 2.8: The illustration of (2.21). One layered neural network

- Choose a suitable optimization algorithm
  - How the optimization process looks
  - Existing optimization algorithms

Optimization part. Backpropagation algorithm

The main goal is getting the numerical solution of the DE and for this aim is to use the residual (2.22) and minimize it over the parameters of the neural network:

$$\min_{A^l, b^l, \dots, A^1, b^1} \mathcal{L} = \min_{A^l, b^l, \dots, A^1, b^1} \mathcal{L} = \frac{1}{|X|} \sum_{x \in X} \|R(x)\|^2$$

Now, how to minimize this complex function? Using the least-squares leads to solving the equations or use gradient-based optimization. For the estimation, the values of the neural network parameters use the gradient-based methods and iteratively goes to the local minimum(!). Suppose, the for the point collocation method randomly choose the set of points at the k-th step, the loss is calculated and gradients are calculated:.

$$\nabla A^l_k = \nabla_{A^l} \mathcal{L}_k, \quad A^l_{k+1} = A^l_k - \lambda(k)\psi(\nabla A^l_k) \tag{2.26}$$

the ψ is the main part of the particular algorithm because using the $\psi(x) = x$, stochastic gradient descent (SGD) immediately have gotten. Using different ψ, the corresponding methods are obtained [32], [10], [19], [9].

Optimizers comparison To demonstrate the quality of various optimization algorithms, a simple neural network architecture was chosen and trained to approximate the function. Lines are the average value of the loss function at a particular iteration, the region of the corresponding color is the region in which the error may lie on average. To collect such statistics, the neural network was trained by each optimizer 25 times. The results are presented in the figure 2.9.

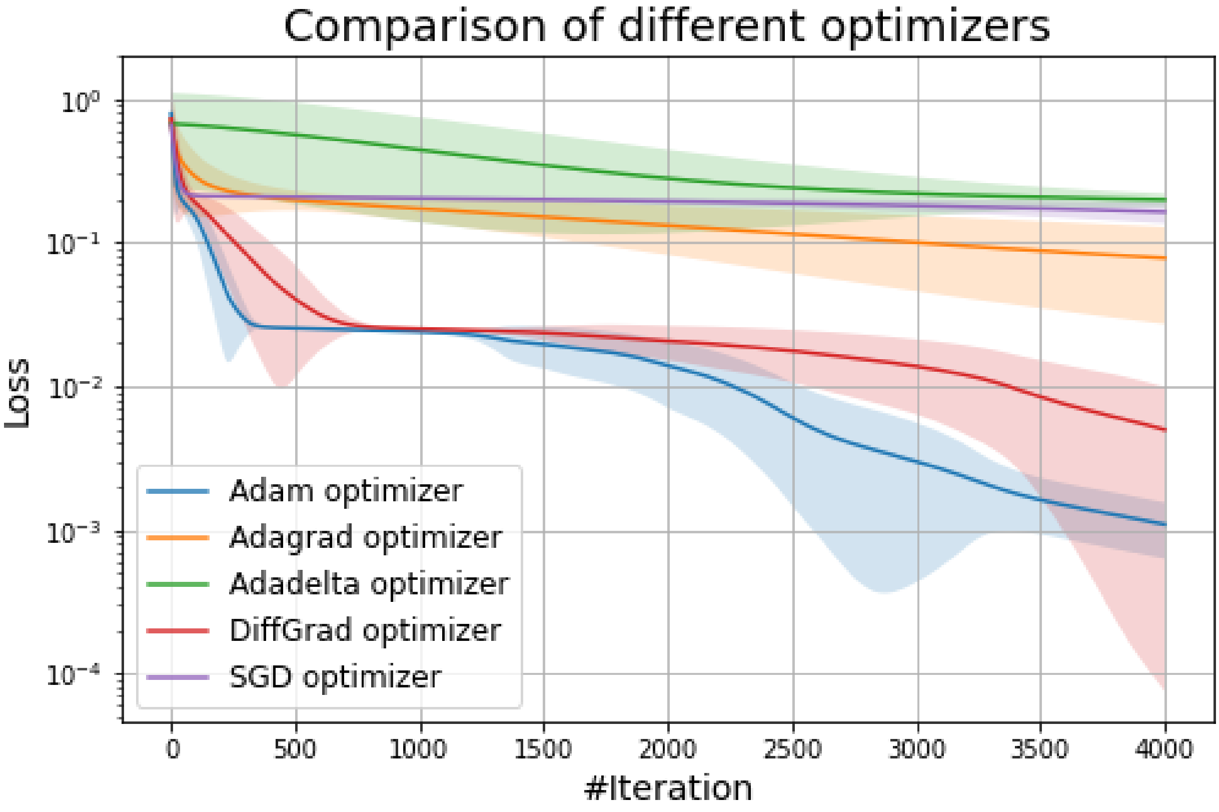

Figure 2.9: Comparison of different optimizers for fixed neural network architecture

Consider the sequence of the operators (2.25) and the quality function or loss function $\mathcal{L}$. Currently not important what loss function and the nature of the operators:

$$\mathcal{N} = A^l \circ \phi^{l-1} \circ A^{l-1} \circ \dots \circ \phi^1 \circ A^1, \quad \mathcal{L} = \mathcal{L}\left(\mathcal{N}\right)$$

For efficient evaluating the gradients over the parameters exists a backpropagation[10] algorithm [7]. The key idea is to use the chain rule for the derivative:

$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial A^l} = \nabla_{A^l}\mathcal{L} \\ \dfrac{\partial \mathcal{L}}{\partial A^{l-1}} = \left[\phi^l\right]' \left[A^{l-1}\right]^T \nabla_{A^l}\mathcal{L} \\ \dfrac{\partial \mathcal{L}}{\partial A^{l-2}} = \left[\phi^{l-1}\right]' \left[A^{l-2}\right]^T \left[\phi^l\right]' \left[A^{l-1}\right]^T \nabla_{A^l}\mathcal{L} = \left[\phi^{l-1}\right]' \left[A^{l-2}\right]^T \dfrac{\partial \mathcal{L}}{\partial A^{l-1}} \\ \text{For k-th derivative in the same way:} \\ \dfrac{\partial \mathcal{L}}{\partial A^{l-k}} = \left[\phi^{l-k+1}\right]' \left[A^{l-k}\right]^T \dfrac{\partial \mathcal{L}}{\partial A^{l-k+1}} \end{cases}$$

[10] In machine learning, backpropagation (backprop, BP) is a widely used algorithm in training feedforward neural networks for supervised learning. Generalizations of backpropagation exist for other artificial neural networks (ANNs), and for functions generally – a class of algorithms referred to generically as ”backpropagation” - from Wikipedia

Now it is known how a neural network works, how it is trained and why a solution can be built with arbitrary accuracy. Next, we will consider different architectures of neural networks for solving different problems, and different approaches, for example, the approach based on the Galerkin method, when the Dirichlet boundary conditions are embedded in a neural network. There is also an approach based on the Ritz method that reduces the solution of the equation to an extremal problem. For example, when solving equations, it is possible to integrate the boundary conditions into the approximator structure [21] [23]. Here the solution is presented in the form:

$$y_h = A(x) + B(x)\mathcal{N}(x) \tag{2.27}$$

where A satisfies the boundary conditions of the first and second kind, where A satisfies the boundary conditions of the first and second kind, and B is in a sense a function of distance, or rather a function that “removes” the values of the model (neural network) at the boundary.

Consider equation $\phi\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}\right) = 0$ and boundary conditions $y(0) = y_0, y(1) = y_1$. In this case, the solution will be built in the form:

$$y_h = (1-x)y_0 + xy_1 + (1-x)x\mathcal{N}(x)$$

Thus, for such a form, a neural network is only part of the solution, for points within a region. It is clear that the name of the complex boundary conditions for the partial differential equation to construct a solution in this form is very difficult. In this form, it is convenient to search for a solution having homogeneous boundary conditions of the first kind. You can use the results from [12], where it is proposed to construct the solution in such a way as to satisfy only conditions of the first kind, and transfer conditions of the second and third kind to the neural network again (to the loss function), example:

$$\phi\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}\right) = 0, \quad y(0) = y_0, \frac{dy}{dx}\bigg|_{x=0} = y_1$$
$$y_h = (1-x)y_0 + B(x)\mathcal{N}(x), \quad \mathcal{L}' = \mathcal{L} + \lambda\left\|\frac{dy_h}{dx}\bigg|_{x=0} - y_1\right\|$$

Another approach [6] also embeds the boundary conditions in the general solution, however, it occurs due to an additional term that estimates the error between the conditions and the solution itself at the boundary and embeds the additional term in a row in order to satisfy the conditions. In fact, every few iterations of the network training, the term is recalculated (a small system of equations is solved) and adjusted to the boundary conditions. Not quite an easy way to implement, however, the quality of the final solution depends on the boundary conditions, on the structure of the additional unit and the necessary accuracy. Models based on the Galerkin method are quite common, so the authors [29] proposed the structure of the model so that, with an increase in the dimension of the problem, the quality of the solution remains acceptable. Their model looks interesting, combines many breakthrough deep learning approaches, but in view of this, the speed of learning is very low. The authors themselves in their work provide an assessment of the training time and the necessary capacities for this - it takes an order of magnitude more time on a conventional personal computer than classical approaches require, but the main goal is high-dimensional tasks, where the algorithm really showed good quality. All approaches proposed and considered below can be divided into 2 groups:

- Embed in the solution itself [21] [23] [6]
- Consider a conditional problem solved by the Lagrange method. In the learning process, the model learns not only to solve the equation itself, but is also fined for not satisfying the boundary conditions [26]

Each group has its own characteristics, so for methods from the first group, the high quality of the solution is characteristic, but the difficulty of drawing up the presentation of the solution is high. The second group is characterized by a not very high quality solution, especially at the borders, however, with sufficient training time and properly selected regularization, this problem is solved, but the plus is the ease of implementation.

For the demonstration of the proposed approach consider the first example - ODE:

$$\frac{dy}{dx} = sin(x), \quad y(0) = -1$$

The loss function for the training neural network:

$$\begin{aligned}
&\mathcal{N} = A^1\sigma\left[A^0x + b^0\right] + b^1 \\
&\qquad\qquad \mathcal{L} = \sum_{x_k \in X}\left[\frac{d}{dx}\mathcal{N}\big|_{x=x_k} - sin(x_k)\right]^2 + \lambda\left[\mathcal{N}(0) + 1\right]^2 \\
&\left(\hat{A}^1, \hat{A}^0, \hat{b}^1, \hat{b}^0\right) = \min_{A^1, A^0, b^1, b^0}\mathcal{L} = \min_{A^1, A^0, b^1, b^0}\sum_{x_k \in X}\left[\frac{d}{dx}\mathcal{N}\big|_{x=x_k} - sin(x_k)\right]^2 + \\
&\qquad\qquad\qquad\qquad + \lambda\left[\mathcal{N}(0) + 1\right]^2
\end{aligned} \tag{2.28}$$

The training process on the figure 2.10 describes the solving process and the values of the loss per training iteration for n independent launches. So, it is important, because in fact, when the neural network created, parameters initialized randomly using algorithms presented by [14] [17] [28]. The comparison of the FDM solution and the neural network solution describes table 2.1. In this work was used Xavier initialization for all networks. For the convenience of the analysis the table 2.1 results presented in figure 2.11. Here it can be seen that the accuracy of the FDM increases

Method	Parameters num	Accuracy
FDM	10	7.8210^{-3}
FDM	25	2.8810^{-3}
FDM	50	1.4410^{-3}
FDM	100	0.6910^{-3}
FDM	200	0.3410^{-3}
ANN	8	0.29810^{-3}
ANN	10	0.11110^{-3}
ANN	20	0.010510^{-3}
ANN	50	0.0041310^{-3}

Table 2.1: Accuracy of the solution for different number of the parameters

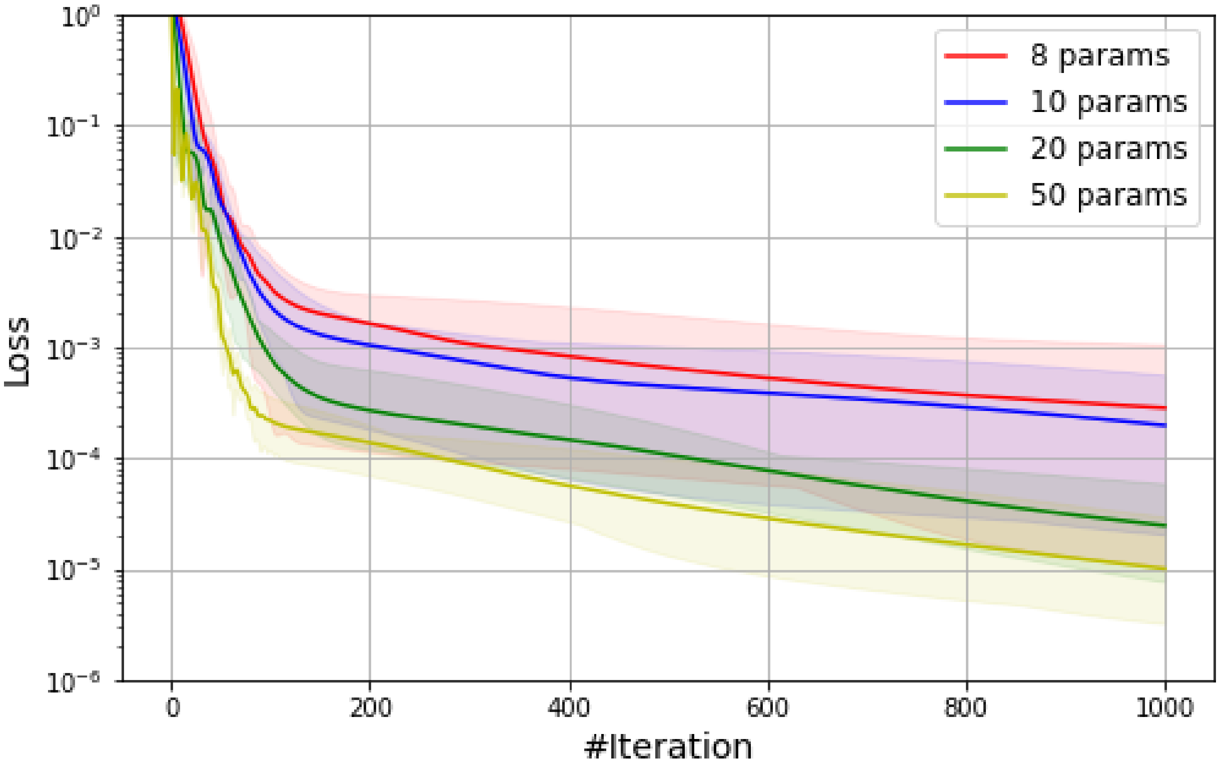

Figure 2.10: Training process for several

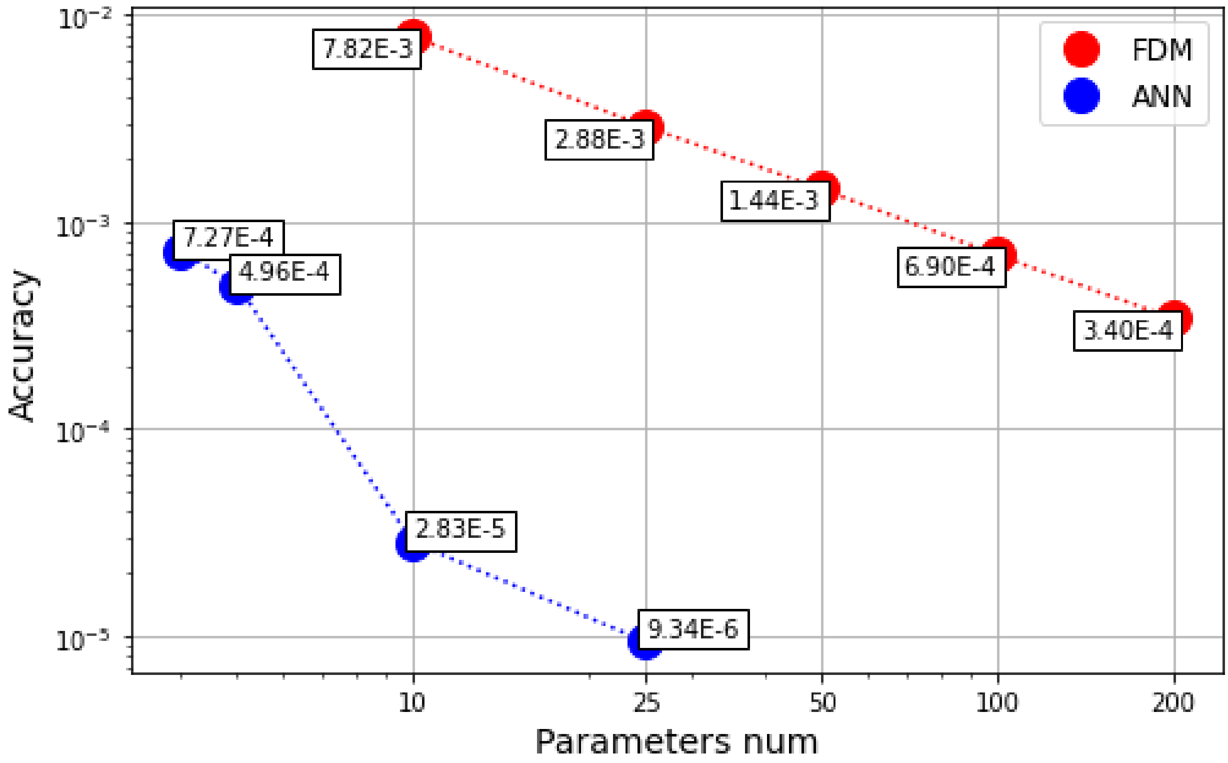

Figure 2.11: Graphical description of the table 2.1

linearly on the logarithm-logarithm plot, which means, that for the decrease the loss function on the 1 order is possible if the number of the parameters will increase in 2 times.

Future analysis will be provided in some steps, first is to apply a neural network-based solver to some number of the ODE's, simple linear, nonlinear, and for the system. The next step includes the application of the solver to single PDE, like the Poisson equation and wave equation, one nonlinear ODE. The last step will be provided for the system's of the PDEs: Stokes equation and Linear elasticity equations. Each of the presented steps will include the comparison of the used number of the parameters for the ANN and for the numerical method, such as FDM or FEM.

2.3.3 The solution of systems of ordinary differential equations (S-ODE)

The solution of a system of ordinary differential equations reduces to the solution of each equation separately, with other functions already known. The idea is to replace each function with a neural network, or use one with several outputs, where each output is a separate function. The loss function will be the sum of the loss function for each equation in the system.

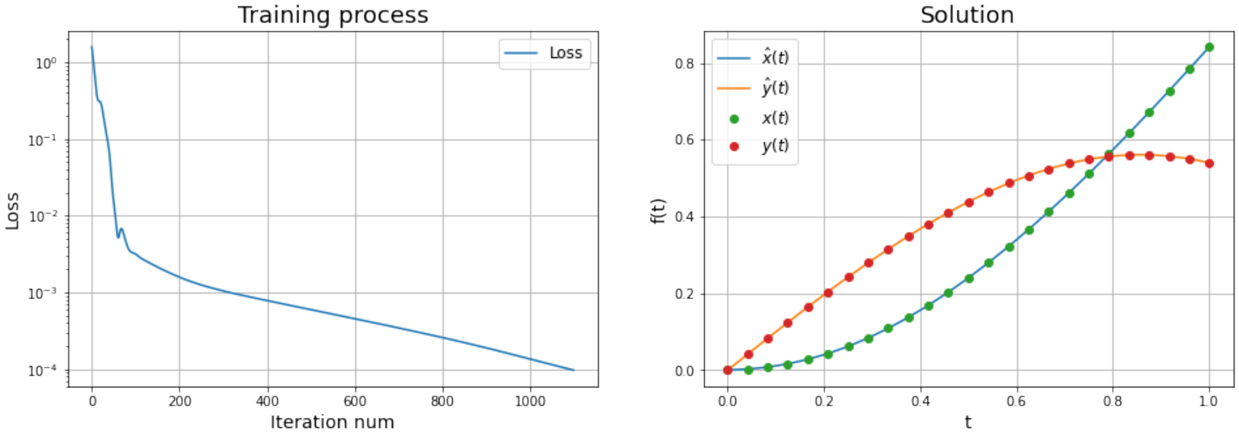

Figure 2.12: Solution of equation (2.29), $\mathcal{L} = 2.14565 \cdot 10^{-7}$

As an example, the system of equations (2.29) will be solved.

$$\begin{cases} \dfrac{dx}{dt} = sin(t) - y \\ \dfrac{dy}{dt} = cos(t) + x \\ x(0) = x_0, y(0) = y_0 \end{cases} \tag{2.29}$$

To apply the proposed approach, it is necessary to formulate the objective function:

$$\begin{gathered} \mathcal{N} = [\mathcal{N}_x, \mathcal{N}_y]^T, \quad \begin{cases} \dfrac{dx}{dt} = sin(t) - y \\ \dfrac{dy}{dt} = cos(t) + x \\ x(0) = x_0, y(0) = y_0 \end{cases} \implies \mathcal{L} = \mathcal{L}_1 + \mathcal{L}_2 + \mathcal{L}_{\text{boundary}} \\ \mathcal{L} = \frac{1}{|T|} \sum_{t \in T \subset R} \left[\frac{d\mathcal{N}_x}{dt} - sin(t) + \mathcal{N}_y \right]^2 + \\ + \frac{1}{|T|} \sum_{t \in T \subset R} \left[\frac{d\mathcal{N}_y}{dt} - cos(t) - \mathcal{N}_x \right]^2 + \\ + \lambda_1 \left[\mathcal{N}_x\right]^2 \big|_{t=0} + \lambda_2 \left[\mathcal{N}_y\right]^2 \big|_{t=0} \end{gathered} \tag{2.30}$$

Figure 2.12 the left part is the process of minimizing the error function, the right part is the result of a solution using a neural network - lines and the analytical solution - circles. It can be seen that the solution coincides, despite the fact that only 18 parameters are used in the neural network.

$f(x) =$	Solution	Equation number
$-2sin(x)cos(y)$	$y = sin(x)cos(y)$	1
$2y^2 + 2x^2$	$y = x^2y^2$	2

Table 2.2: Examples of equations to consider

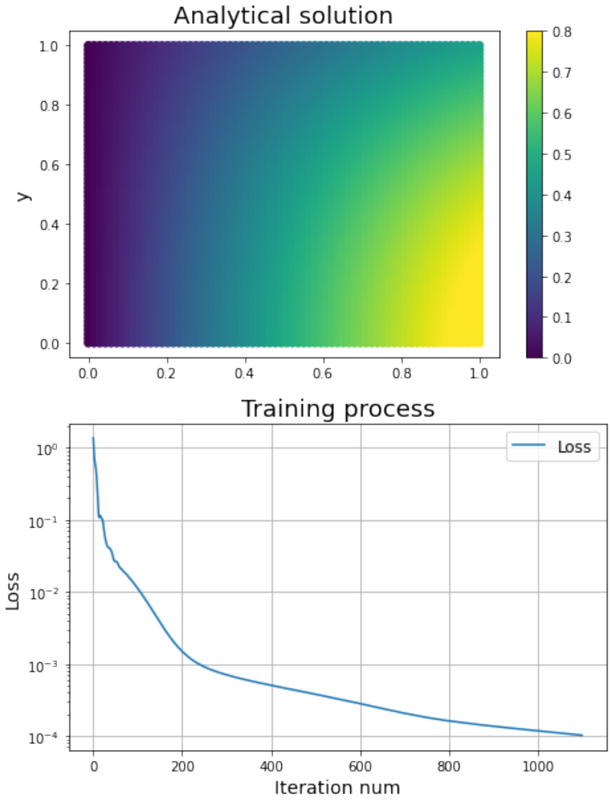

Figure 2.13: Solution of equation 1 from the table 2.2

2.3.4 The solution of partial differential equations (PDE)

To solve partial differential equations, the same approach is used as for solving ordinary differential equations. By example, the Poisson equation will be solved.

$$\nabla^2 u = f(x), \quad \nabla^2 = \nabla \cdot \nabla, \forall x \in \Omega$$

The results of solving each equation are presented in graphs 2.13, 2.14. To solve the equations, we considered a network structure including 1 one with a sigmoidal activation function. The process of training a neural network took about 10 seconds, which is quite a long time, but it is important to understand that the number of parameters is 18, which is very small for construction and the convergence to the solution is slow because of this.

In table 2.3, the accuracy column reflects the average normalized root of the deviation, or in other words, the average deviation from the analytical solution. The figures show that the solution built by the neural network is close to true, and also reflects all the features of the function. The convergence on the graphs without problems and jumps between local extremes, which is good.

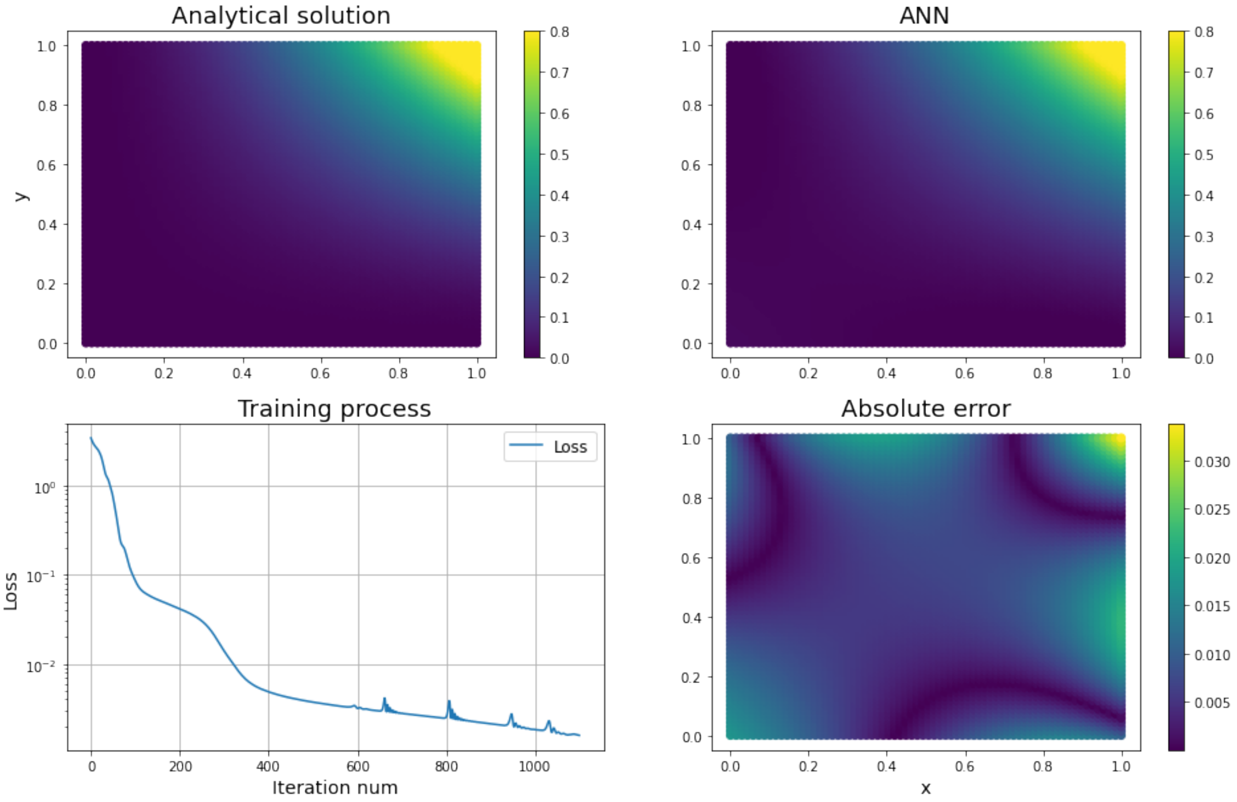

Figure 2.14: Solution of equation 2 from the table 2.2

$f(x) =$	Solution	Equation number	Solution accuracy	Figure
$-2sin(x)cos(y)$	$y = sin(x)cos(y)$	1	6.0114e-06	2.13
$2y^2 + 2x^2$	$y = x^2y^2$	2	7.1142e-05	2.14

Table 2.3: Results for the equations from the table 2.2

In general, to solve the specifically given equation, the method is suitable for any other too, but it may require adjustment of the network architecture, the choice of an optimizer and the duration of the training.

Conclusions

In this chapter, approximation problems based on linear regression were considered. For linear regression, basic methods for estimating coefficients were considered, and problems that may arise in the process of their estimation were also considered, such as insufficiently generalized ability due to lack of data or strong collinearity of data, leading to a high value of the condition number. Then, from the basic task of supervised learning, a transition was made to replacing the kernel in linear regression and possible replacements, such as the transformation of linear regression into the restoration of the image of the function in the Fourier and Chebyshev space, were considered. Specifically, these functions were considered in view of the fact that there are strong theorems guaranteeing convergence in pointwise and in the sense of the L_2 norm. Also, theorems on decreasing coefficients cannot be left unnoticed, since without loss of quality it is possible to limit these expansions in the future and not worry about subsequent effects. After the conclusions made that in the general case, with an arbitrary core, the linear regression model only builds an expansion in basic functions, it was suggested that a single-layer neural network is a similar expansion, with a pre-selected core - a sigmoid function. For this function, there is Tsybenko's theorem guaranteeing the quality of approximation of arbitrary accuracy with an increase in the number of terms in the expansion. All these conclusions lead to the conclusion that to solve differential equations it remains only to correctly compose the optimization problem, then the solution will be guaranteed to be found, if it exists for the initial problem itself. Ideas and conclusions were borrowed from various works, and are indicated in the list of sources. However, it is worth noting that no work with a similar approach has been encountered before and this issue will be further worked out. In addition it is worth noting the second fact that the current chapter fully explains that to solve differential equations by the method using neural networks, it is enough to use single-layer networks, which significantly narrows the family of architectures for learning. An increase in depth is guaranteed to lead to an improvement in the solution with sufficient training time, however, effects appear associated with jumps in the objective function during training process due to the large number of local minima. For the tasks, we used the same optimization algorithm based on gradient descent and the influence of the algorithm parameters, as well as the choice of the algorithm itself on the quality of the solution and the rate of convergence, was not investigated, but most likely the results will differ and still this is important. At the end, solutions of typical problems for ODEs, system of ODE, and a one-dimensional partial differential equation were presented. Partial differential equations systems are presented in a next chapter, since in addition to the introduced criteria for training a neural network, an important feature of the work is the solution of partial differential equations systems.

Chapter 3

Numerical results

This chapter will focus on solving systems of differential equations, such as the Stokes equations and the linear elasticity problem. To assess the quality of the solution of the Stokes problem, a comparison will be made with analytically the velocity profile equation. An assessment of the solution of the linear elasticity problem will be carried out with the result of the obtained finite element method. The finite element solution was to build using free Fenics software. An analysis will be made of the quality of the solution regarding the architecture of neural networks, the number of parameters and activation functions.

3.1 Stokes equations

The Stokes equation or, more generally, the Navier-Stokes equation describes a fluid flow. The required functions in the equation are the vector field of fluid velocities and the scalar pressure field. The difficulty in solving this problem lies in the fact that the equations themselves are nonlinear, which immediately indicates the use of iterative methods for solving the resulting system of algebraic equations. More than that, the solution of such equations reduces to finding the Jacobi matrix, or even the Hessian matrix. These matrices are relatively easy to find and work with, in the case where the number of equations and variables is small, but in practice using non-trivial methods of constructing a discrete space leads to hundreds of thousands of variables. Another point complicating an already difficult decision in computer terms is the amount of necessary resources for storing matrices, for example, the Jacobi matrix will require quadratically proportional number of cells with respect to the number of variables, and Hesse cubically depends. If it is possible to rearrange the order of the equations, it is possible to achieve some structure of the matrix such that it is effectively stored and / or processed. All these difficulties lead to an increase in the time complexity of solving the problem, this is not counting the fact that the selection of a high-quality grid and the repeated solution of the problem are required.

So, the system of equations is as follows:

$$\begin{cases} (u\cdot\nabla)\,u = \nabla^2 u - \nabla p \\ \nabla\cdot u = 0 \\ u = \vec{u} = [u_1, u_2]^T = [u_x, u_y]^T \end{cases} \tag{3.1}$$

, where $\nabla^2 = \nabla\cdot\nabla$. The more detailed explanation is here [30], [27].

First, we rewrite the equation in component form:

$$\begin{cases} u\cdot\nabla u = \nabla^2 u - \nabla p \\ \\ \nabla\cdot u = 0 \end{cases} = \begin{cases} u_i\dfrac{\partial u_i}{\partial x_i} = \dfrac{\partial^2 u_i}{\partial x_i^2} - \nabla p_i, \quad \forall i \in \{1,2\} \\ \sum_i \dfrac{\partial u_i}{\partial x_i} = 0 \end{cases} \tag{3.2}$$

Boundary conditions will be considered later. The problem is considered immediately in a dimensionless form, since it is required to stabilize the solution and reduce the number of degrees of freedom of the equation being solved. As before, let us now consider an arbitrary neural network with several outputs, each of which will correspond to a specific desired function:

$$\begin{aligned} \vec{\mathcal{N}} = [\mathcal{N}_{u_1}, \mathcal{N}_{u_2}, \mathcal{N}_p] = A^l \circ \phi^{l-1} \circ A^{l-1} \circ \cdots \circ \phi^1 \circ A^1 = \\ = A^l\left[\phi^{l-1}\left[\dots\left[A^1(x) + b^1\right]\dots\right] + b^{l-1}\right] + b^l \end{aligned} \tag{3.3}$$

, where

$$\begin{gathered} \vec{n} = n_0, \dots, n_l, \quad n_1 = 2, n_l = 3 \\ A^1 \in R^{n_0\times n_1}, A^2 \in R^{n_1\times n_2} \dots, A^k \in R^{n_k\times n_{k+1}} \\ b^1 \in R^{n_1}, b^2 \in R^{n_2} \dots, b^k \in R^{n_k} \end{gathered} \tag{3.4}$$

For simplicity, one can describe a neural network by a sequence of numbers n_i, such that the first term sequentially characterizes the dimension of the space on which the equation is solved, and the last characterizes the required number of outputs or the number of required functions. Now the neural network can be written in compact form:

$$\begin{aligned} \mathcal{N}_{\vec{n}} = A^l_{n_l\times n_{l-1}} \circ \phi^{l-1} \circ A^{l-1}_{n_{l-1}\times n_{l-2}} \circ \cdots \circ \phi^1 \circ A^1_{n_0\times n_1} = \\ = A^l_{n_l\times n_{l-1}}\left[\phi^{l-1}\left[\dots\left[A^1_{n_0\times n_1}(x) + b^1_{n_1}\right]\dots\right] + b^{l-1}_{n_{l-1}}\right] + b^l_{n_l} \\ A^l_{n_l\times n_{l-1}} \in R^{n_l\times n_{l-1}}, b^{l-1}_{n_{l-1}} \in R^{n_{l-1}} \end{aligned} \tag{3.5}$$

Now, in fact, if a specific value of l is fixed, we can say that the space of parameters of a neural network can be considered as the Cartesian product of the subspaces of the corresponding parameters:

$$W_A = \prod_{k=1}^{l} R^{n_k \times n_{k+1}}, \quad \{A^1, \dots, A^l\} \in W$$

$$W_b = \prod_{k=1}^{l} R^{k}, \quad \{b^1, \dots, b^l\} \in W_b$$

Such simplifications were introduced for ease of recording the conditions for conducting numerical experiments.

To solve the problem, as was done before, it is necessary to introduce an objective function that will be optimized:

$$\begin{cases} u_1 \dfrac{\partial u_1}{\partial x_1} = \dfrac{\partial^2 u_1}{\partial x_1^2} + \dfrac{\partial^2 u_1}{\partial x_2^2} - \nabla p_1 \\ u_2 \dfrac{\partial u_2}{\partial x_2} = \dfrac{\partial^2 u_2}{\partial x_1^2} + \dfrac{\partial^2 u_2}{\partial x_2^2} - \nabla p_2 \\ \dfrac{\partial u_1}{\partial x_1} + \dfrac{\partial u_2}{\partial x_2} = 0 \end{cases} \tag{3.6}$$

We will carry out several operations sequentially, to begin with, substitute the neural network in the equation and calculate the derivatives. And also immediately introduce the residual for each equation:

$$\begin{cases} R_1 = \mathcal{N}_{u_1} \dfrac{\partial \mathcal{N}_{u_1}}{\partial x_1} - \dfrac{\partial^2 \mathcal{N}_{u_1}}{\partial x_1^2} - \dfrac{\partial^2 \mathcal{N}_{u_1}}{\partial x_2^2} + \nabla \mathcal{N}_{p_i} \\ R_2 = \mathcal{N}_{u_2} \dfrac{\partial \mathcal{N}_{u_2}}{\partial x_2} - \dfrac{\partial^2 \mathcal{N}_{u_2}}{\partial x_1^2} - \dfrac{\partial^2 \mathcal{N}_{u_2}}{\partial x_2^2} + \nabla \mathcal{N}_{p_2} \\ R_3 = -\dfrac{\partial \mathcal{N}_{u_1}}{\partial x_1} - \dfrac{\partial \mathcal{N}_{u_2}}{\partial x_2} \end{cases} \tag{3.7}$$

, where R_i - residual for equation i. The key idea is to minimize the residual vector and find the optimal parameters:

$$\vec{R} = [R_1, R_2, R_3]^T, \quad W_A^*, W_b^* = \text{arg} \min_{W_A, W_b} \left[R \cdot R^T \right] \tag{3.8}$$

W - a set of optimal parameter values for model 3.3 in form (3.1).

Table 3.1 describes all configurations to consider. Problem (3.1) is considered inside a rectangular region, with boundary conditions:

$$\begin{cases} u_x = u_y = 0, \quad y \in \{0, 1\} \\ p = 0.1, \quad x = 0 \\ p = 0.0, \quad x = 1 \end{cases} \tag{3.9}$$

In the table 3.1 the conditions of the first experiment are described, which includes the use of several simple architectures to approximate the solution. It is important to understand how large

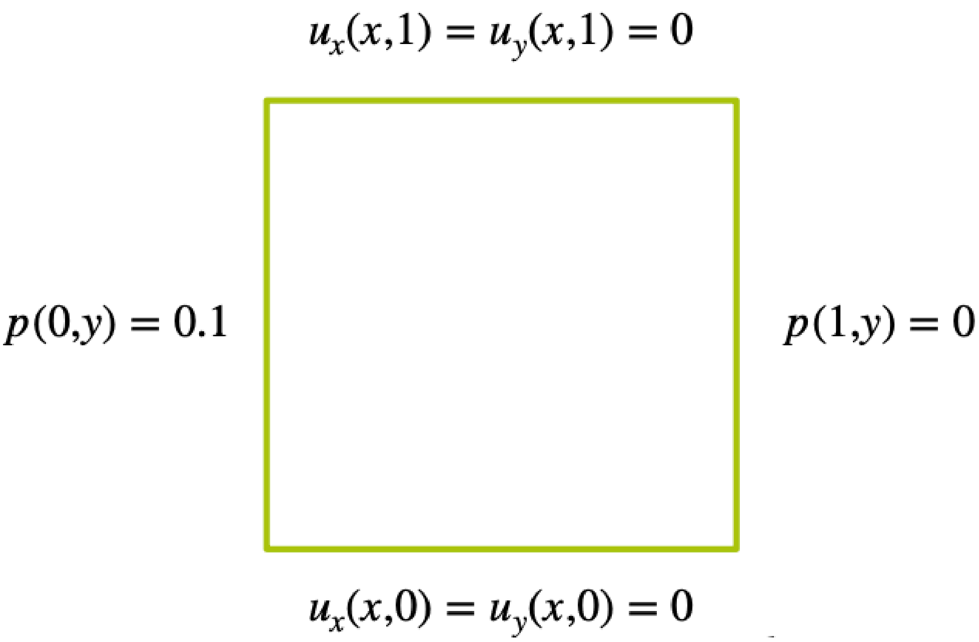

Figure 3.1: Ω and boundary conditions from (3.13)

$\vec{n}$	Parameters number	Accuracy
$[2,4,3]$	20	0.00489
$[2,8,3]$	40	0.00053
$[2,16,3]$	70	0.00021
$[2,4,4,3]$	36	0.00195
$[2,8,8,3]$	104	0.00022
$[2,16,16,3]$	334	0.00016
$[2,4,4,4,3]$	52	0.00076
$[2,8,8,8,3]$	168	0.00016
$[2,16,16,16,3]$	592	0.00011

Table 3.1: Accuracy of the solution for different number of the parameters [Stokes equation]

a configuration is needed to achieve a certain accuracy. Further, after choosing the architecture, we can talk about using other activation functions to analyze the behavior of the solution.

From table 3.1 it is seen that an increase in the number of parameters in the model leads to an improvement in accuracy almost always, with some caveats: it does not make sense to increase the width of the layers, it makes no sense to increase the number of layers when their width is small.

An important result of table 3.1 and figure 3.5 is the fact that it is now clear that the optimal architecture for such an equation should not include more than 3 layers of width 8. What does this mean? Answer: the number of parameters greater than 104 with an architecture of $\mathcal{N}_{[2,8,8,3]}$ is optimal. Further, the question will be posed differently: is it possible with an already fixed architecture using activation functions like cosine or Chebyshev polynomials to build an even more accurate solution.

The learning process was depicted for all architectures, as this would lead to a misunder-

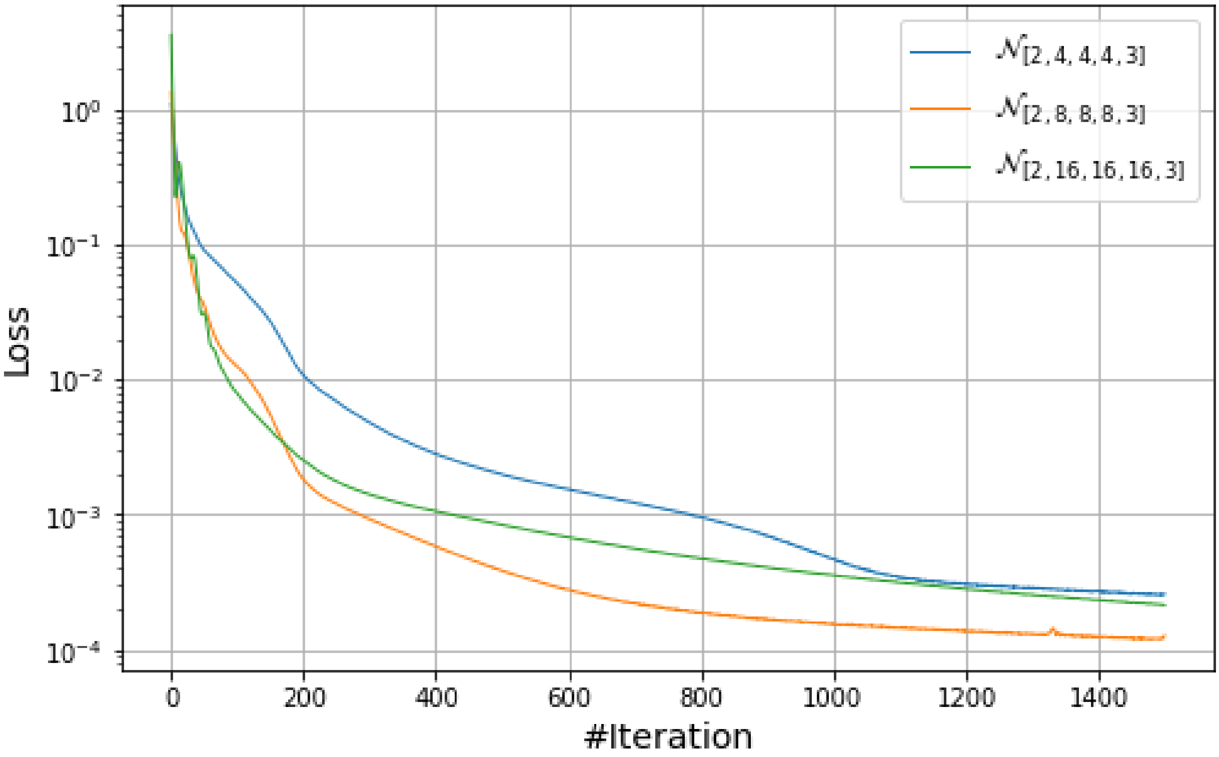

Figure 3.2: Training process for ANNs from table 3.1

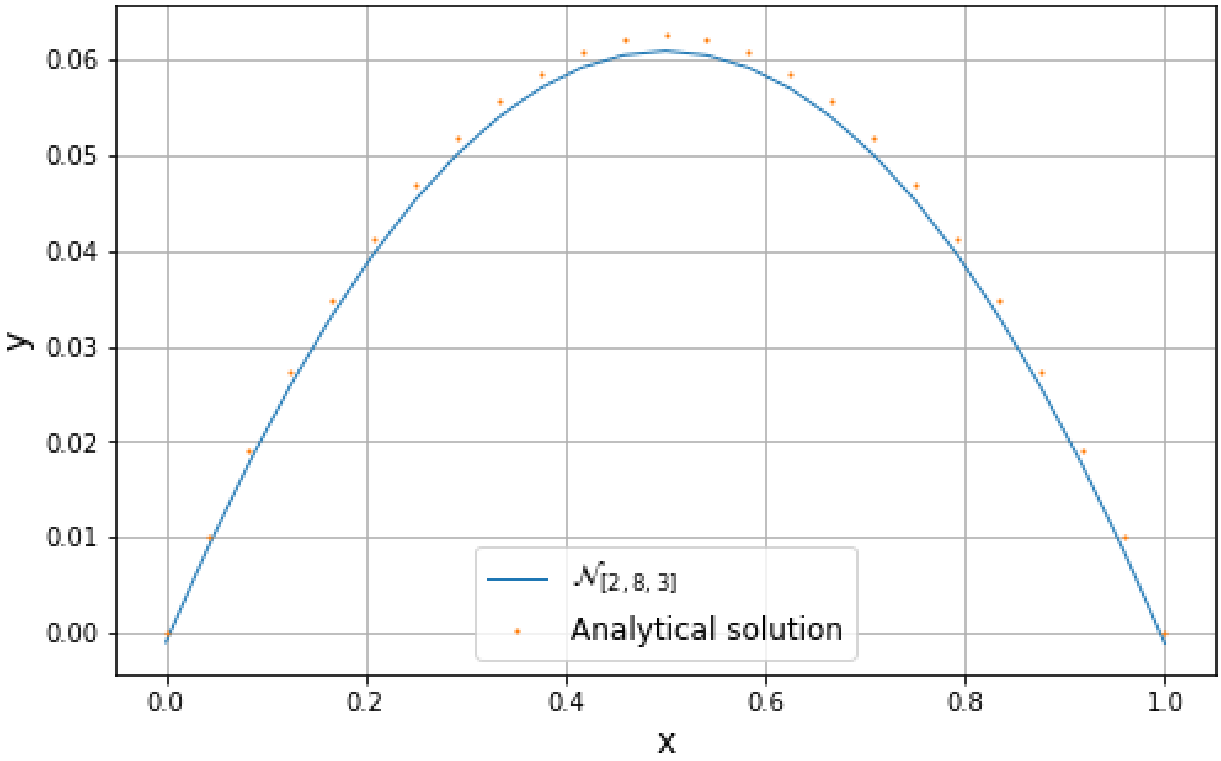

Figure 3.3: Velocity profile for ANNs from table 3.1

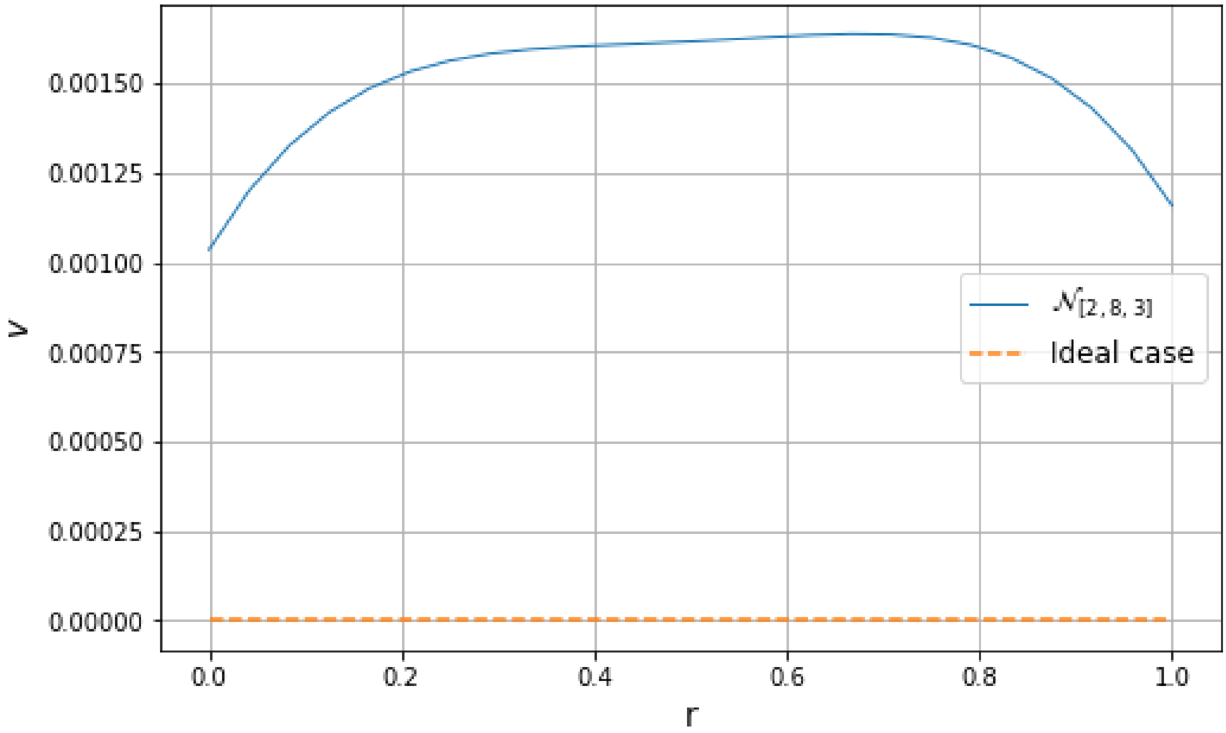

Figure 3.4: Velocity profile error for ANNs from table 3.1

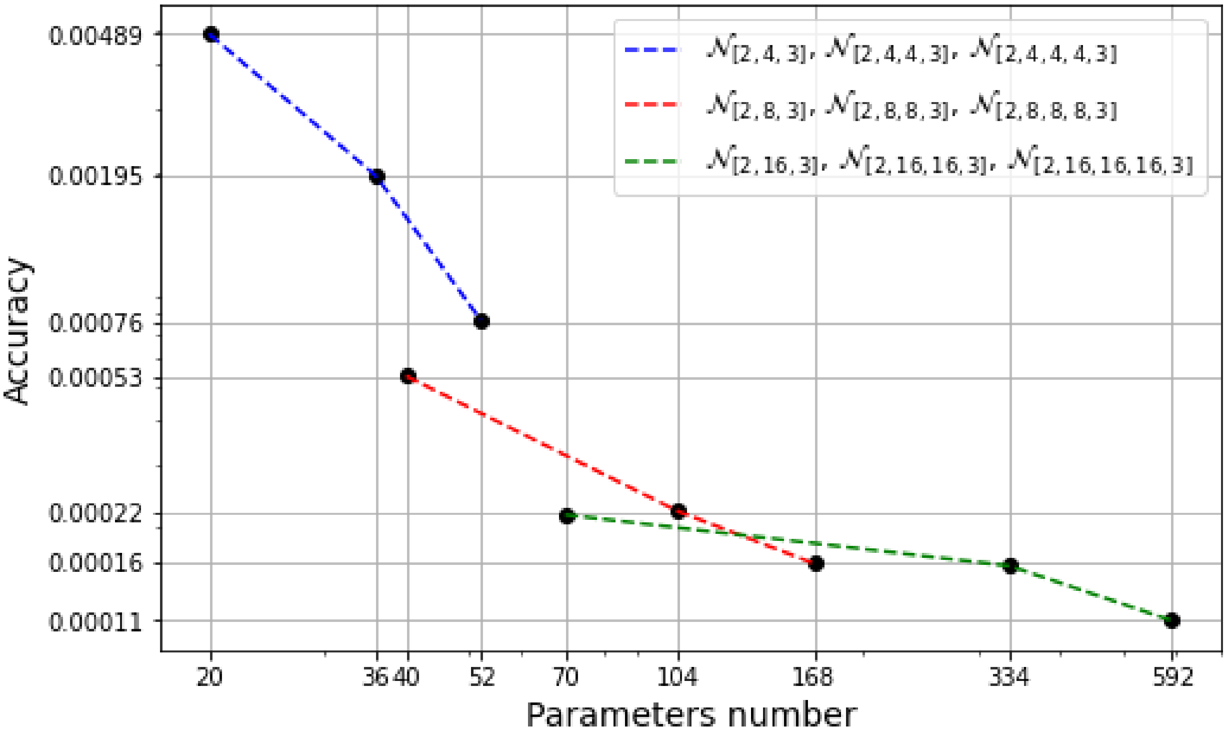

Figure 3.5: Parameters number vs Accuracy, description of the table 3.1

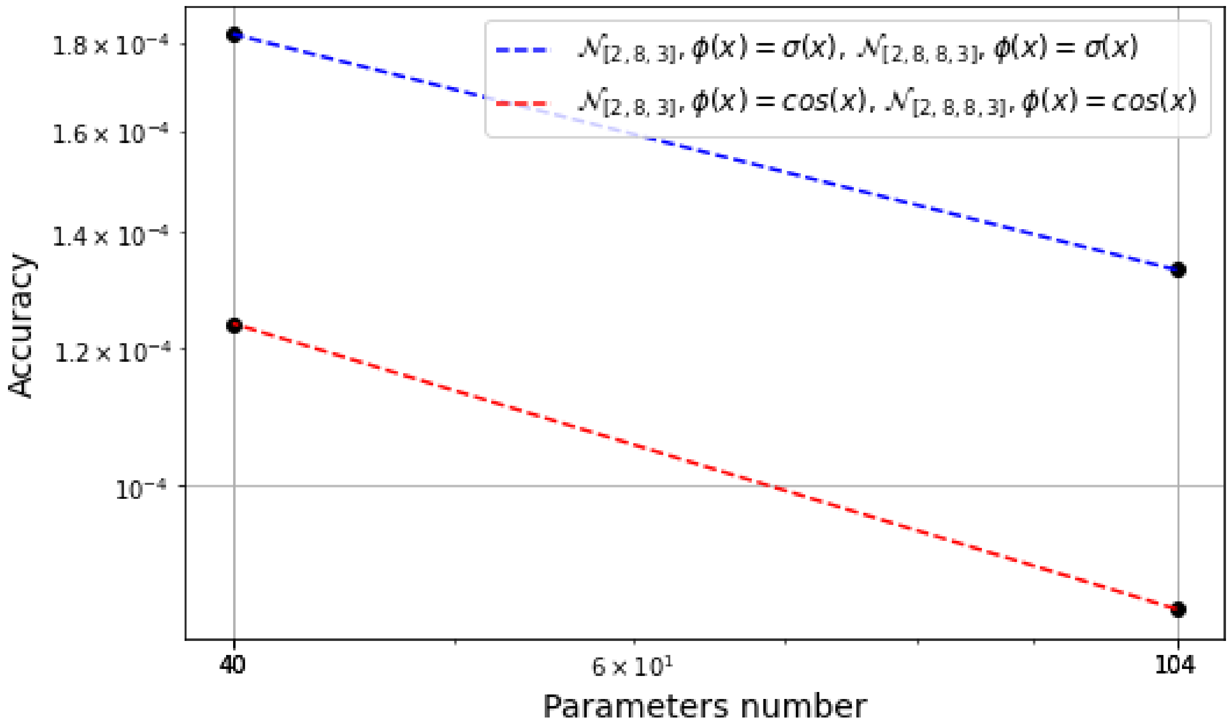

Figure 3.6: Parameters number vs Accuracy, description of the table 3.1

standing of the drawing due to its overcrowding. Also, an important result is that the architecture using the cosine activation function gave an increase compared to the usual sigmoid function.

Having chosen about the optimal architecture of the neural network to solve the equation, we can compare whether the improvement of the approximation ability gives the replacement of the activation function within a fixed network configuration. Figure 3.6 shows that there is a change in quality for the better, the calculation was carried out more than 10 times for each model in order to collect statistically significant results. The analytical form of the solution obtained for the figure, respectively:

$$\mathcal{N}^{\sigma}_{[2,8,3]} = A^1\sigma\left[A^0x + b^0\right] + b^1$$

$$\mathcal{N}^{cos}_{[2,8,3]} = A^1 cos\left[A^0x + b^0\right] + b^1$$

It is important to remember that not only the replacement of the activation function was used, but also the criterion (2.10), (2.11) corresponding to it, which guaranteed the orthogonality of the functions obtained on the last layer:

$$\mathcal{L}_{\text{regularization}} = \left[\frac{sin(W - W^T) + B - B^T}{2W - 2W^T} + \frac{sin(W + W^T) + B + B^T}{2W + 2W^T} - I\right]_F$$

3.2 Linear elasticity equations

Linear elasticity is a mathematical model of the behavior of solids and describes the internal stress state under the influence of small deformations. A simplified model of a more complete theory of nonlinear elasticity and a section on the mechanics of solid bodies. The main assumption is that the deformations are small and the generalized Hooke's law applies. This problem was chosen to demonstrate the method, since it is often used in conjunction with filtration models, and poroelastic media models are used. If it is possible to solve the proposed problem with high accuracy of approximation, then it will be possible to use this approach to simulate more complex processes, such as hydraulic fracturing.

The system of equations for the linear elasticity problem is as follows:

$$\begin{cases} -\nabla\sigma = f \\ \sigma = \lambda\epsilon_v I + 2\mu\epsilon, \quad \epsilon_v = Tr\left(\epsilon\right) \\ \epsilon = \dfrac{1}{2}\left[\nabla u + \left(\nabla u\right)^T\right] \\ u = \vec{u} = \left[u_x, u_y\right]^T, \quad \vec{x} = (x, y) \in \Omega \subset R^2 \end{cases} \tag{3.10}$$

More detailed explanation available here [2], [15]. In component form, the system of equations looks like this:

$$\nabla u = \begin{pmatrix} \dfrac{\partial}{\partial x} \\ \dfrac{\partial}{\partial y} \end{pmatrix} \begin{pmatrix} u_x & u_y \end{pmatrix} = \begin{pmatrix} \dfrac{\partial}{\partial x}u_x & \dfrac{\partial}{\partial x}u_y \\ \dfrac{\partial}{\partial y}u_y & \dfrac{\partial}{\partial y}u_y \end{pmatrix}, -\nabla\sigma = -\nabla\left[\lambda\epsilon_v I + 2\mu\epsilon\right] =$$

$$= \lambda \begin{pmatrix} \dfrac{\partial}{\partial x}\dfrac{\partial}{\partial x}u_x + \dfrac{\partial}{\partial x}\dfrac{\partial}{\partial y}u_y \\ \dfrac{\partial}{\partial y}\dfrac{\partial}{\partial x}u_x + \dfrac{\partial}{\partial y}\dfrac{\partial}{\partial y}u_y \end{pmatrix} + \mu \begin{pmatrix} 2\dfrac{\partial}{\partial x}\dfrac{\partial}{\partial x}u_x + \dfrac{\partial}{\partial y}\dfrac{\partial}{\partial y}u_y + \dfrac{\partial}{\partial y}\dfrac{\partial}{\partial x}u_y \\ \dfrac{\partial}{\partial x}\dfrac{\partial}{\partial x}u_y + \dfrac{\partial}{\partial x}\dfrac{\partial}{\partial y}u_y + 2\dfrac{\partial}{\partial y}\dfrac{\partial}{\partial y}u_y \end{pmatrix} \Longrightarrow \tag{3.11}$$

$$\begin{cases} \lambda\left[\dfrac{\partial}{\partial x}\dfrac{\partial}{\partial x}u_x + \dfrac{\partial}{\partial x}\dfrac{\partial}{\partial y}u_y\right] + \mu\left[2\dfrac{\partial}{\partial x}\dfrac{\partial}{\partial x}u_x + \dfrac{\partial}{\partial y}\dfrac{\partial}{\partial y}u_y + \dfrac{\partial}{\partial y}\dfrac{\partial}{\partial x}u_y\right] = -f_x \\ \lambda\left[\dfrac{\partial}{\partial y}\dfrac{\partial}{\partial x}u_x + \dfrac{\partial}{\partial y}\dfrac{\partial}{\partial y}u_y\right] + \mu\left[\dfrac{\partial}{\partial x}\dfrac{\partial}{\partial x}u_y + \dfrac{\partial}{\partial x}\dfrac{\partial}{\partial y}u_y + 2\dfrac{\partial}{\partial y}\dfrac{\partial}{\partial y}u_y\right] = -f_y \end{cases}$$

And so, the system of equations is described. The boundary conditions are as follows:

$$\vec{n} \cdot \vec{u} = U, \forall\vec{x} \in \partial\Omega_u, \quad \vec{n} \cdot \sigma = \vec{T}, \forall\vec{x} \in \partial\Omega_T \tag{3.12}$$

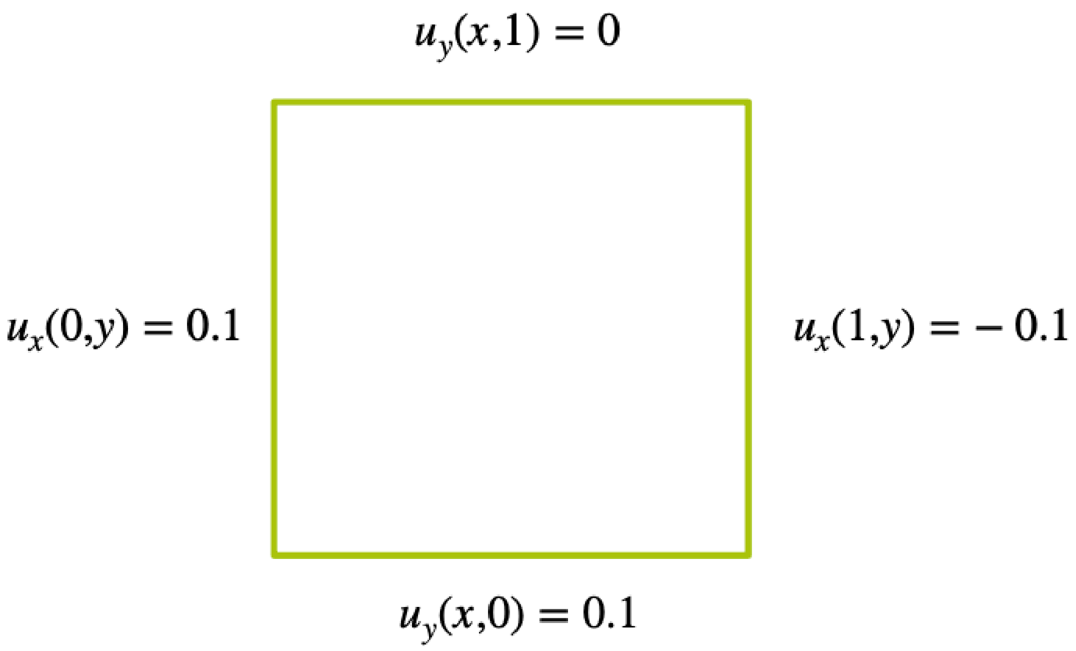

Figure 3.7: Ω and boundary conditions from (3.13)

$\vec{n}$	Parameters number	Accuracy
$[2, 4, 3]$	20	0.00021
$[2, 8, 3]$	40	0.00007
$[2, 16, 3]$	70	0.00002
$[2, 4, 4, 3]$	36	0.00016
$[2, 8, 8, 3]$	104	0.000014
$[2, 16, 16, 3]$	334	0.00001

Table 3.2: Accuracy of the solution for different number of the parameters [Linear elasticity]

Similarly, as for the Stokes equations, computational experiments will be carried out for different architectures. A study will also be conducted on the effect of the activation function on the quality of the solution. Model architectures will be defined as (3.5), table 3.2 describes all configurations to consider. Problem (3.10) is considered inside a rectangular region, with boundary conditions:

$$\begin{cases} u_x = 0.1, \quad x = 0 \\ u_x = -0.1, \quad x = 1 \\ u_y = 0.1, \quad y = 0 \\ u_y = 0.0, \quad y = 1 \end{cases} \tag{3.13}$$

Table 3.2 describes the structures of the models on which the experiment will be conducted. To assess the optimal structure, figure 3.8 will be used, showing the results of table 3.2.

The solution to problem (3.10) is the function u, which describes the field of displacements of elementary volumes, the description [2].

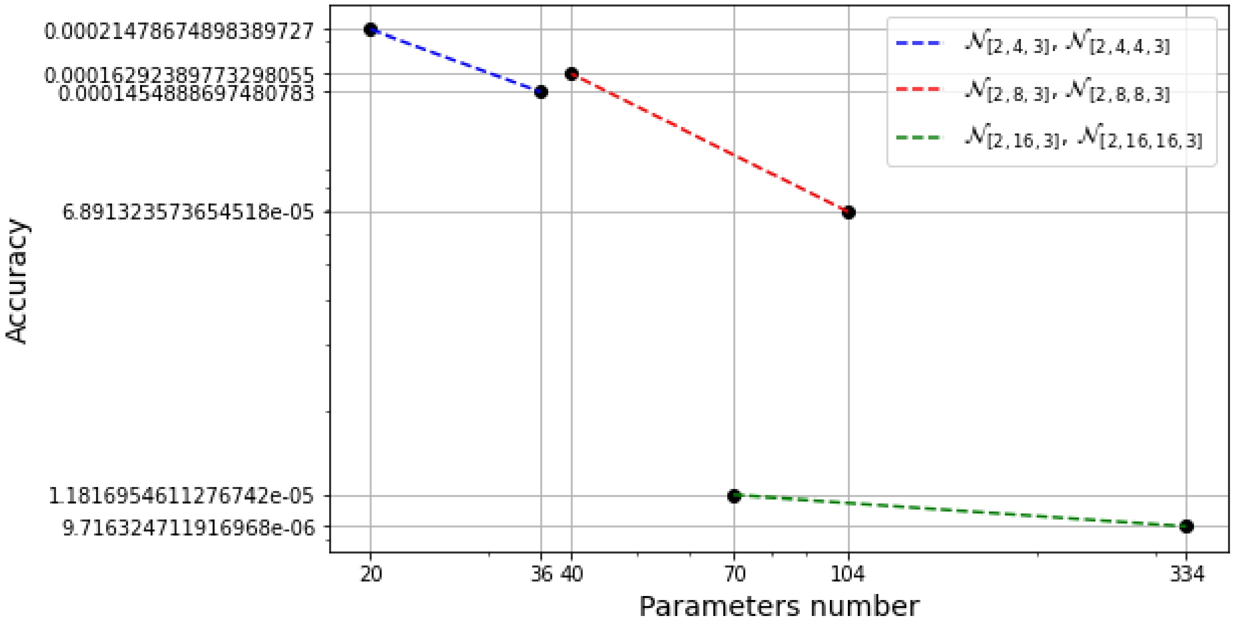

Figure 3.8: Parameters number vs Accuracy, description of the table 3.2

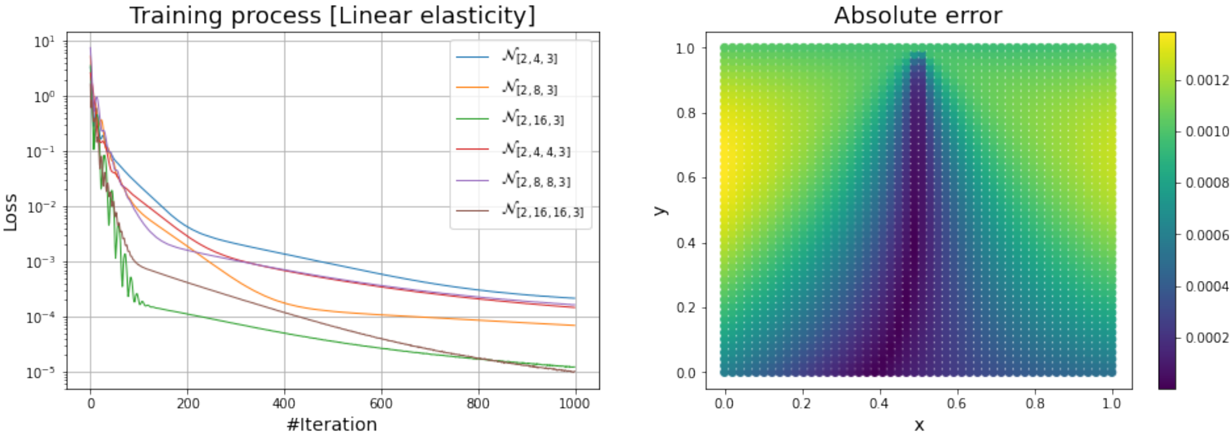

Figure 3.9: Parameters number vs Accuracy, description of the table 3.2

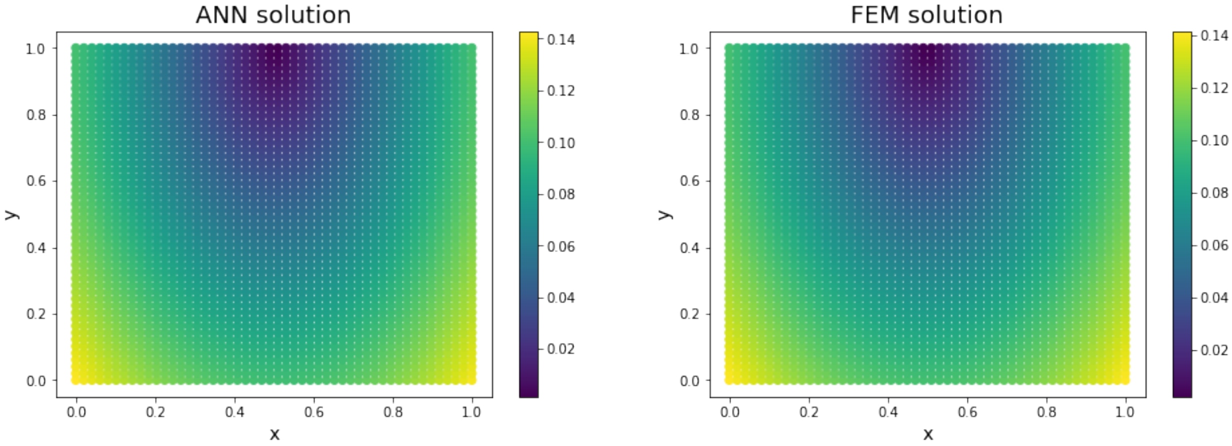

Figure 3.10: Solution for the (3.10)

It can be seen from figure 3.8, 3.10 that an increase in depth does not lead to such a strong solution quality as an increase in the network width, moreover, in terms of the computational complexity of the algorithm, it becomes clear that it is optimal to use no more than 2-3 layers, then the quality increase for the required computational costs getting small. In fact, it's not particularly important to try to use other activation functions, it's important that for the given tasks it's already becoming clear that the work describing neural networks with hundreds or even thousands of parameters is exhaustively large for such tasks, since it is clear that for systems of equations that although they are solved analytically, they are still quite complicated. It is also worth noting the connection between the availability of an analytical solution and the required number of parameters for solving problems.

Conclustions

In this chapter, 2 problems were solved, the linear elasticity equation, which describes the deformations and stresses that arise under certain actions on the region, and the Stokes problem, which describes the fluid flow in a narrow channel. Both problems are described by systems of partial differential equations, linear and nonlinear. For each task, several different configurations of neural networks were considered. All networks were successfully trained, the quality and result of which greatly depends on the number of parameters, as well as their order. An important conclusion can be considered the fact that it makes no sense to use an arbitrary configuration, which will often be too exhaustive, and the required training time is too long. We could dwell on this conclusion, however, the treatability of the result remains another important issue, therefore, in Chapter 2, we considered possible approaches that would consider neural networks as an expansion of functions in some series. For this, 2 specific series, the Fourier series and the Chebyshev series are considered in sufficient detail. So, substituting the activation function of the cosine and the specially obtained criterion, one can obtain expansion in the Fourier series.

Chapter 4

Conclusions

4.1 Applicability

4.2 Further work

4.3 Discussion of the results

Appendix A

Solution of non-autonomous differential equations

Appendix B

Details of the implementation of the construction and training of neural networks

Bibliography

[1] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions: With Formulas, Graphs, and Mathematical Tables*. Applied mathematics series. Dover Publications, 1965.

[2] J.R. Barber. *Elasticity*. Solid Mechanics and Its Applications. Springer, 1992.

[3] C.M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer, 2006.

[4] Maxime Bocher. *Introduction to the Theory of Fourier's Series*. Mathematics Department, Princeton University, Annals of Mathematics, Second Series, Vol. 7, No. 3, 2020.

[5] Terence Candes, Emmanuel; Tao. The dantzig selector: Statistical estimation when p is much larger than n. pages 2313—2351, 2007.

[6] Linlin Cao, Ran He, and Bao-Gang Hu. Locally imposing function for generalized constraint neural networks - a study on equality constraints, 2016.

[7] Y. Chauvin and D.E. Rumelhart. *Backpropagation: Theory, Architectures, and Applications*. Developments in Connectionist Theory Series. Taylor & Francis, 2013.

[8] I. Dimov, I. Faragó, and L. Vulkov. *Finite Difference Methods. Theory and Applications: 7th International Conference, FDM 2018, Lozenetz, Bulgaria, June 11-16, 2018, Revised Selected Papers*. Lecture Notes in Computer Science. Springer International Publishing, 2019.

[9] Shiv Ram Dubey, Soumendu Chakraborty, Swalpa Kumar Roy, Snehasis Mukherjee, Satish Kumar Singh, and Bidyut Baran Chaudhuri. diffgrad: An optimization method for convolutional neural networks, 2019.

[10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.

[11] B.A. Finlayson. *The Method of Weighted Residuals and Variational Principles*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 2013.

[12] C.A.J. Fletcher. *Computational Galerkin Methods*. Scientific Computation. Springer Berlin Heidelberg, 2012.

[13] J.E. Gentle. *Matrix Algebra: Theory, Computations, and Applications in Statistics*. Springer Texts in Statistics. Springer, 2007.

[14] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[15] P.L. Gould, P.L. Gould, and P. L. *Introduction to Linear Elasticity*. Springer New York, 1983.

[16] S.S. Haykin. *Neural Networks: A Comprehensive Foundation*. International edition. Prentice Hall, 1999.

[17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[18] Arthur E. Hoerl; Robert W. Kennard. Ridge regression: Biased estimation for nonorthogonal problems. pages 55—67, 1970.

[19] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[20] R. Kress. *Numerical Analysis*. Graduate Texts in Mathematics. Springer New York, 2012.

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

[22] Ji Zhu Li Wang, Michael D. Gordon. Regularized least absolute deviations regression and an efficient algorithm for parameter tuning. pages 690—700, 2006.

[23] Zeyu Liu, Yantao Yang, and Qing-Dong Cai. Solving differential equation with constrained multilayer feedforward network, 2019.

[24] J.C. Mason and D.C. Handscomb. *Chebyshev Polynomials*. CRC Press, 2002.

[25] Weijie Su Emmanuel J. Candes Małgorzata Bogdan, Ewout van den Berg. Statistical estimation and testing via the ordered l1 norm. 2013.

[26] G. P. Purja Pun, R. Batra, R. Ramprasad, and Y. Mishin. Physically informed artificial neural networks for atomistic modeling of materials. *Nature Communications*, 10(1), May 2019.

[27] M. Rieutord. *Fluid Dynamics: An Introduction*. Graduate Texts in Physics. Springer International Publishing, 2014.

[28] Andrew M Saxe, James L McClelland, and Surya Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120*, 2013.

[29] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, Dec 2018.

[30] R. Temam. *Navier-Stokes equations: theory and numerical analysis*. Studies in mathematics and its applications. North-Holland Pub. Co., 1979.

[31] Robert Tibshirani. Regression shrinkage and selection via the lasso. pages 267—288, 1996.

[32] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012.

[33] С.Ю. Городецкий and В.А. Гришагин. *Нелинейное программирование и многоэкстремальная оптимизация: учеб. пособие*. Модели и методы конечномерной оптимизации. Изд-во Нижегор. госуниверситета, 2007.