LUT UNIVERSITY
LUT School of Energy Systems
LUT Mechanical Engineering

Denis Bobylev

IMPLEMENTATION OF ARTIFICIAL INTELLIGENCE APPROACHES IN THE FIELD OF MULTIBODY DYNAMICS USING KERAS

Examiner(s): Professor Aki Mikkola
Dr. Sc. Tech. Grzegorz Orzechowski

Master's Thesis 2019

ABSTRACT

Lappeenranta University of Technology
LUT School of Energy Systems
LUT Mechanical Engineering

Denis Bobylev

Implementation of Artificial Intelligence approaches in the field of Multibody Dynamics using Keras

Master’s Thesis

2019

65 pages, 35 figures, 4 tables and 1 appendix

Examiners: Professor Aki Mikkola
Dr. Sc. (Tech.) Grzegorz Orzechowski

Keywords: multibody dynamics, nonlinear system, artificial intelligence, neural networks, machine learning, Python, Keras, MatLab.

The aim of the research was to develop an approach of utilizing of neural networks in the field of multibody dynamics using Python and its Artificial Intelligence library Keras. The hypothesis of the research is corresponding to the fact that the utilizing of neural networks and Artificial Intelligence is capable to bring the field of Multibody Dynamics to the new level by increasing the performance, decreasing setup time, that results the development of meta-model that can be used for a predictive design, data analysis and design optimization.

In the practical part of the project software such as Matlab, Python was used. At the first stage, the experimental data for two nonlinear functions was edited and saved as HDF5 file using Python and Matlab. Then the obtained file was used as the dataset for training of Neural Network that was built using Keras. At the second stage the feasibility of the approach was tested on the example of double pendulum. First, the equations of motion were obtained using MatLab, then the Neural Network was developed for the system fed by the data obtained from the simulation of equations of motion. At the next stage the networks were tested and the results were analyzed in order to get the results of the research.

Results of the research are described using graphs and tables that demonstrate the performance of neural networks for two studied cases: system of two nonlinear equations and double pendulum connected with torsional spring.

ACKNOWLEDGEMENTS

I would like to express gratitude and thank LUT University for the opportunity to pass through the Master's program at this university, my family, my girlfriend and my friends for constant support and inspiration. Their contribution to success cannot be overestimated.

Moreover, I want to warmly thank Professor Aki Mikkola for the chance to work in this exciting project under his leadership. Without your passion, guidance and motivation it would be challenging to succeed in this research. Also, I would like to thank Dr. Sc. Tech. Grzegorz Orzechowski for his help, valuable advices and proposals that helped me to open the World of Artificial Intelligence.

Denis Bobylev

Denis Bobylev

Lappeenranta 10.08.2019

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

Latin symbols

$\mathbf{A}_i$	Rotation matrix at body *i*
$\dot{\mathbf{A}}_i$	Derivative of the rotation matrix at body *i*
a, a + 1	Layers of Neural Network
b	Bias
g	Gravity constant
$\bar{\mathbf{G}}_i$	Local velocity transformation matrix between angular velocities and first derivative of Euler parameter of body *i*
$\dot{\bar{\mathbf{G}}}_i$	Derivative of local velocity transformation matrix between angular velocities and first derivative of Euler parameter of body *i*
$\mathbf{I}$	Identity matrix
I_1	Moment of inertia of link I
I_2	Moment of inertia of link II
k	Spring coefficient
L	Lagrangian
$\mathbf{M}$	Mass matrix
m_1	Mass of link I
m_2	Mass of link II
l_1	Length of link I
l_2	Length of link II
n	Number of predictions
$\mathbf{Q}_e$	Vector of generalized forces
$\mathbf{Q}_v$	Vector of quadratic velocity
$\mathbf{q}$	Vectors of generalized coordinates
$\mathbf{q}_i$	Vectors of generalized coordinates of body *i*
$\dot{\mathbf{q}}_i$	Vectors of generalized velocities of body *i*
$\ddot{\mathbf{q}}$	Vectors of generalized accelerations
$\mathbf{R}_i$	Position of body reference coordinate system relative to global coordinate system *XYZ*

$\dot{\mathbf{R}}_i$	Velocity of body reference coordinate system
$\ddot{\mathbf{R}}_i$	Acceleration of body reference coordinate system
$\mathbf{r}_{iP}$	Vectors of position of point *P* at body *i* in global system
$\dot{\mathbf{r}}_{iP}$	Vectors of velocity of point *P* at body *i* in global system
$\ddot{\mathbf{r}}_{iP}$	Vectors of acceleration of point *P* at body *i* in global system
T	Kinetic energy
t	Time
$\bar{\mathbf{u}}_{iP}$	Vectors of position of point *P* with at body *i* respect to body reference coordinate
$\tilde{\bar{\mathbf{u}}}_{iP}$	Skew symmetric matrix of position vector of point *P*
V	Potential energy
w_1, w_{n_x}	Weights
W_{iner}	Work of inertia forces
W_{ext}	Work of external forces
$x_1, x_2, x_3,$ x_4, x_5, x_6, x_{n_x}	Inputs
x_{1p}, x_{2p}	Positions of masses in *X* coordinate
$\dot{x}_{1p}, \dot{x}_{2p}$	Velocities of masses in *X* coordinate
y	Output
y_{1p}, y_{2p}	Positions of masses in *Y* coordinate
y_i	Actual value
y_i^p	Predicted value
$\dot{y}_{1p}, \dot{y}_{2p}$	Velocities of masses in *Y* coordinate

Greek symbols

α_1, α_2	Angles of link I and link II
$\dot{\alpha}_1, \dot{\alpha}_2$	Derivatives of angles
$\boldsymbol{\theta}_{iE}$	Rotational Euler parameters
$\dot{\boldsymbol{\theta}}_{iE}$	First time derivative of rotational Euler parameters
$\ddot{\boldsymbol{\theta}}_{iE}$	Second time derivative of rotational Euler parameters
$\theta_0, \theta_1, \theta_2, \theta_3$	Euler parameters
$\bar{\boldsymbol{\omega}}_i$	local angular velocity of body *i*

$\tilde{\tilde{\boldsymbol{\omega}}}_i$	Skew-symmetric matrix of local angular velocity of body *i*

Abbreviations

0D	Zero-Dimensional
2D	Two-Dimensional
3D	Three-Dimensional
4D	Four-Dimensional
5D	Five-Dimensional
ADAM	Adaptive Moment Estimation Algorithm
AI	Artificial Intelligence
API	Application Programming Interface
DL	Deep Learning
DNN	Deep Neural Network
DOF	Degrees of Freedom
LSTM	Long Short Term Memory Neural Network
MAE	Mean Absolute Error
MAPE	Mean Absolute Percentage Error
ML	Machine Learning
MSE	Mean Squared Error
MSLE	Mean Squared Log Error
NN	Neural Network
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Decent

1 INTRODUCTION

Nowadays, multibody modeling is an important tool and method in the design process of mechatronic systems. Following the current global trend in the field of machines that aim to become more energy efficient, powerful, lightweight and reliable, the power of computational resources combined with numerical calculations have increased the interest and involvement of computer simulations in the machine design development.

The goal of multibody dynamics is to learn the movement of mechanical systems that have a number of rigid or flexible bodies connected with the joints, constraints or force components. Those forces can have different sources and vary by complexity. Dynamics of such systems is dealing with the principles used in classical mechanics. These abilities of multibody dynamics are commonly used for investigations in medical, robotics and computer games applications. (Flores 2015, p. 1.)

At the same time, machine learning (ML) has been gaining the interest as the tool that can be utilized for the range of different problems. It is a tool which is widely introduced in different areas and industries. It has become one of the most fascinating and interesting topics for those industries that are aimed at efficient processing of the data in order to learn the patterns of the data and bring it to the new level of understanding and interpretation. A distinctive feature of this technology is the fact that machine is learning how to perform task for which it was not explicitly preprogrammed. These aspects define the applications where the use of ML is justified: image and speech processing, medical diagnosis, classification, regression, prediction, statistical arbitrage etc. (Salvaris et al. 2018, p. 1.)

In the field of mechanical engineering ML can be useful while designing, simulating and analyzing the product. It can be considered as a good tool for improving the performance of designed machine by proceeding different sources of the data, such as experimental and empirical data. As the result, ML is a tool which is able to learn this data and using the examples make the conclusion about possible drawbacks and potential problems of the system. These facts open up potential opportunities for further development and integration of ML in the field of mechanical engineering and similar problems where these techniques and approaches can be utilized.

Using ML, it is possible to develop meta-model that is able to learn and analyze the input data automatically. To enable this, it is necessary to design and develop a neural network (NN) that would be fed and trained by big data which is a combination of experimental data, computer-aided engineering data and empirical data. After processing the data, it is possible to get a meta- model which could be used further for predictive design, data analysis and design optimization. Clarification of this concept is shown in Figure 1.

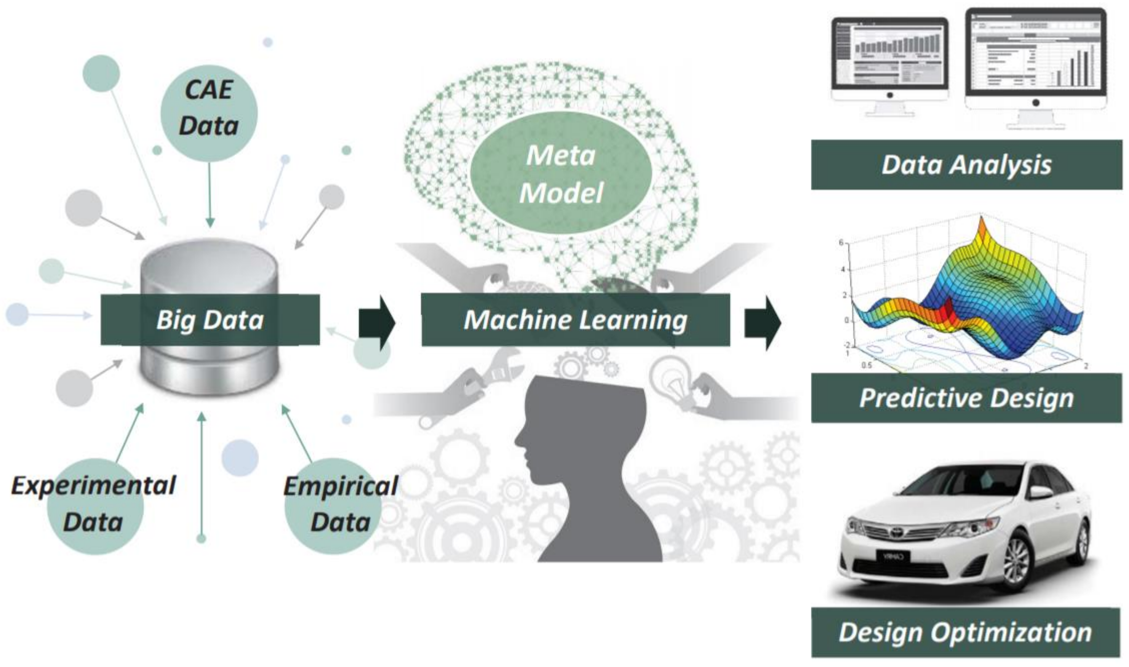


Figure 1. Process flow (Choi 2019).

1.1 Research Objectives

The main task of this thesis work is to find out if the approaches dealing with the artificial intelligence (AI) are able to replace traditional methods used in the field of multibody dynamics.

For this purpose, the first objective is to design and develop a NNs for a non-linear system with six variables consisting of two functions. The main challenge in this part is to define the optimal configuration of the Neural Network that would proceed the input data and show a good performance while testing it.

The second objective is to investigate if the same approach can be used for processing the data obtained from the simulation of double pendulum with a torsional spring. The main aspect is to make sure that the output is close enough to the real system and at the same time avoid overfitting.

Both NNs will be configurated and tested using Keras in Python. They should be developed consequently according to the methods previously used for the similar cases. Then it is necessary to evaluate and analyze obtained NNs models, justify the configurations and the whole performances of the system by comparing them to the real systems and make the conclusion.

1.2 Research questions

There are several research questions considered in this thesis project. The main question is: How is possible to utilize Artificial Intelligence capabilities instead traditional approaches in multibody dynamics?

The following sub-questions stem from the main question of the thesis:

1. How to design and build the NN for the systems under consideration?
2. How accurate are the obtained NNs configurations?
3. How long it takes to setup the NNs?
4. What is the computational efficiency of the obtained systems?

1.3 Thesis structure

The thesis follows the common academic paper structure. It has six chapters: introduction, theoretical background, research methods, results, discussion and conclusion. In the introduction the general information, motivation, research questions and objectives outlined.

Theoretical background covers the knowledge and information used in the thesis obtained from scientific sources. It has the basic information about the Artificial Intelligence, Machine Learning, how it can be utilized in Python using Keras.

Research methods chapter covers the designing and setting aspects of two NNs developed during the research. Moreover, it has the information about software used for in the research.

Results chapter is used for analysis of the results of two Neural Networks built for considered systems obtained from the previous chapter. Discussion describes how obtained results can be utilized, what are possible improvements of networks can be done, which aspects should be considered further and in the following researches. Conclusion is used for summarizing of the thesis, also, it gives the answers to the research questions.

2 THEORETICAL BACKGROUND

This chapter presents the theory that was used to conduct the study. It is focused on the theory of dynamics of multi-body systems, as well as on the theory of Artificial Intelligence, Machine Learning and Neural Networks in general.

2.1 Introduction to Multibody System Dynamics

Multibody system can be recognized as a mechanical system consisting of several interconnected bodies that can be rigid or flexible. The multibody system is characterized by the number of mechanical components that are subjected to translational or rotational displacements and kinematic joints connecting the bodies and restricting their motion in some way. (Flores 2015, p. 1.)

The body can be defined as rigid when the deformation is small and insignificant to the motion done by the body. In this case the body is free to change the position, but the shape stays constant. At the same time, the flexible body is able to change the shape because of elasticity. (Flores 2015, p. 1.)

In the real life the body cannot be categorized as absolutely rigid, but in many cases the bodies are stiff enough, meaning that their flexibility can be neglected. This fact significantly simplifies the modelling process of multibody system, because in this case the motion of the body can be represented by applying six generalized coordinates with six Degrees of Freedom (DOF). If the body is considered to be flexible, it has six DOF and the number of generalized coordinates the are used to express the deformations. (Flores 2015, p. 1.)

The joints presented in multibody system limit the relative motion between the components. The force elements are describing the internal forces existing in the system and related to the relative motion of the components. The forces acting om the multibody system components are associated with the presence of springs, dampers, actuators and external forces. Those forces are used to represent the connections between the parts of the system and the surroundings. (Flores 2015, p. 2.)

The motional equations of the multibody system can be obtained by applying numerical time integration method. First, the kinematics of the multibody system should be defined. Then the dynamic equilibrium of the system can be obtained by applying constraint motional equations based on the principle of virtual work. The kinematics of the system is presented using global formulation. In this case the coordinates of the components are determined with respect to the global frame of reference. (Flores 2015, p. 2.)

2.1.1 Kinematics

Rigid body *i* is presented in Figure 2. It is presented using global coordinate system *XYZ.* The global position $\mathbf{r}_{iP}$ of the point P located inside the rigid body *i* can be described using body reference coordinate system. (Baharudin 2016, p.24.)

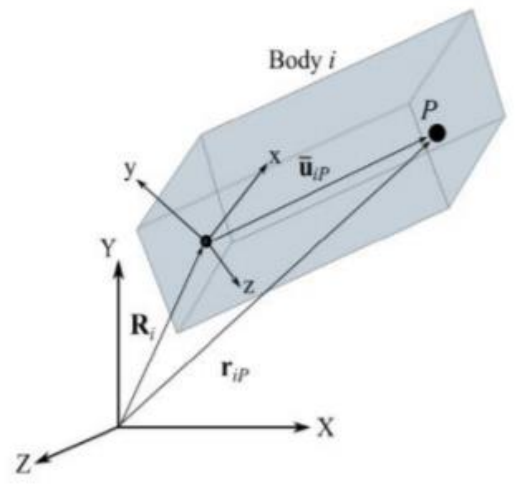


Figure 2. Point P located on rigid body *i* (Baharudin, 2016).

The equation which represents the global position of point P is shown below:

$$\mathbf{r}_{iP} = \mathbf{R}_i + \mathbf{A}_i\bar{\mathbf{u}}_{iP} \quad (1)$$

Where, $\mathbf{R}_i$ shows the position of the body reference system relative to global coordinate system *XYZ,* $\mathbf{A}_i$ is the rotation matrix that defines the orientation of body reference coordinate system relatively to *XYZ,* $\bar{\mathbf{u}}_{iP}$ shows the position of P relatively to reference coordinate system. (Baharudin 2016, p.24.)

Then the generalized coordinates $\mathbf{q}_i$ of the rigid body can be expressed as:

$$\mathbf{q}_i = \left[\mathbf{R}_i{}^{\mathrm{T}} \ \ \boldsymbol{\theta}_{iE}^{\mathrm{T}}\right]^{\mathrm{T}} \tag{2}$$

Where, $\mathbf{R}_i = [\,\mathrm{R}_{iX} \quad \mathrm{R}_{iY} \quad \mathrm{R}_{iZ}]^{\mathrm{T}}$ represents the origin of reference coordinate, $\boldsymbol{\theta}_{iE} = [\,\theta_0 \quad \theta_1 \quad \theta_2 \quad \theta_3\,]^{\mathrm{T}}$ is the rotational coordinates. (Baharudin 2016, p.25.)

$\theta_0\,, \theta_1\,, \theta_2\,, \theta_3$ are Euler parameters. Using these parameters, the rotation matrix $\mathbf{A}_i$ is defined as:

$$\mathbf{A}_i = \begin{bmatrix} 1 - 2\theta_2^2 - 2\theta_3^2 & 2(\theta_1\,\theta_2 \ + \theta_0\,\theta_3\,) & 2(\theta_1\,\theta_3 \ + \theta_0\,\theta_2) \\ 2(\theta_1\,\theta_2 \ + \theta_0\,\theta_3\,) & 1 - 2\theta_1^2 - 2\theta_3^2 & 2(\theta_2\theta_3 \ + \theta_0\,\theta_1\,) \\ 2(\theta_1\,\theta_3 \ + \theta_0\,\theta_2\,) & 2(\theta_2\,\theta_3 \ + \theta_0\,\theta_1\,) & 1 - 2\theta_1^2 - 2\theta_2^2 \end{bmatrix} \tag{3}$$

Applying Euler parameters, it is necessary to follow the next constraint:

$$\boldsymbol{\theta}_{iE}^{\mathrm{T}}\boldsymbol{\theta}_{iE} - 1 = 0 \tag{4}$$

To get the velocity of *P* the equation (1) should be differentiated by time. Then it is expressed as:

$$\dot{\mathbf{r}}_{iP} = \dot{\mathbf{R}}_i + \dot{\mathbf{A}}_i\bar{\mathbf{u}}_{iP} \tag{5}$$

$$\dot{\mathbf{r}}_{iP} = \dot{\mathbf{R}}_i - \dot{\mathbf{A}}_i\tilde{\bar{\mathbf{u}}}_{iP}\bar{\boldsymbol{\omega}}_i \tag{6}$$

Where $\dot{\mathbf{R}}_i$ is the first derivative of the body reference coordinate system, $\dot{\mathbf{A}}_i$ the first derivative of the rotation matrix, $\tilde{\bar{\mathbf{u}}}_{iP}$ is a skew-symmetric matrix of $\bar{\mathbf{u}}_{iP}$, $\bar{\boldsymbol{\omega}}_i$ is the angular speed. (Baharudin 2016, p.26.)

Then $\bar{\boldsymbol{\omega}}_i$ can be expressed as:

$$\bar{\boldsymbol{\omega}}_i = \bar{\mathbf{G}}_i\dot{\boldsymbol{\theta}}_{iE} \tag{7}$$

Where $\bar{\mathbf{G}}_i$ is a local transformation matrix related to local and global component of body *i*, $\dot{\boldsymbol{\theta}}_{iE}$ is the first derivative of vector of Euler parameters. (Baharudin 2016, p.26.)

$\bar{\mathbf{G}}_i$ can be expressed as:

$$\bar{\mathbf{G}}_i = \begin{bmatrix} -\theta_1 & \theta_0 & \theta_3 & -\theta_2 \\ -\theta_2 & -\theta_3 & \theta_0 & \theta_1 \\ -\theta_3 & \theta_2 & -\theta_1 & \theta_0 \end{bmatrix}_i \tag{8}$$

Then $\dot{\mathbf{r}}_{iP}$ can be expressed relatively to $\dot{\mathbf{q}}_i = [\dot{\mathbf{R}}_i \quad \dot{\boldsymbol{\theta}}_{iE}]^{\mathrm{T}}$ as:

$$\dot{\mathbf{r}}_{iP} = [\mathbf{I} \quad -\mathbf{A}_i\tilde{\bar{\mathbf{u}}}_{iP}\bar{\mathbf{G}}_i]\begin{bmatrix} \dot{\mathbf{R}}_i \\ \dot{\boldsymbol{\theta}}_{iE} \end{bmatrix} \tag{9}$$

Where, $\mathbf{I}$ is expressed as 3x3 identity matrix. Then the acceleration vector $\ddot{\mathbf{r}}_{iP}$ is calculated by derivation of equation (9) by time in the next way:

$$\ddot{\mathbf{r}}_{iP} = [\mathbf{I} \quad -\mathbf{A}_i\tilde{\bar{\mathbf{u}}}_{iP}\bar{\mathbf{G}}_i]\begin{bmatrix} \ddot{\mathbf{R}}_i \\ \ddot{\boldsymbol{\theta}}_{iE} \end{bmatrix} + \begin{bmatrix}\mathbf{0} & \mathbf{A}_i\tilde{\bar{\boldsymbol{\omega}}}_i\tilde{\bar{\mathbf{u}}}_{iP}\dot{\bar{\mathbf{G}}}_i\end{bmatrix}\begin{bmatrix} \dot{\mathbf{R}}_i \\ \dot{\boldsymbol{\theta}}_{iE} \end{bmatrix} \tag{10}$$

Where, generalized acceleration $\ddot{\mathbf{q}}_i = [\ddot{\mathbf{R}}_i \quad \ddot{\boldsymbol{\theta}}_{iE}]$, $\ddot{\mathbf{R}}_i$ is the second derivative of the body reference coordinate system, $\ddot{\boldsymbol{\theta}}_{iE}$ is the second derivative of Euler parameters, $\tilde{\bar{\boldsymbol{\omega}}}_i$ is the skew-symmetric matrix of angular velocity, $\dot{\bar{\mathbf{G}}}_i$ is the first derivative of transformation matrix. (Baharudin 2016, p.26.)

2.1.2 Equation of Motion

The motional equation of multibody system can be obtained by applying the principle of virtual work and least motion. After equalizing the works done by external and inertial forces, the dynamic equilibrium for the unconstrained system can be found. It can be expressed as:

$$\delta W_{iner} = \delta W_{ext} \tag{11}$$

Where, δW_{iner} replicates the amount of work done by inertial forces and δW_{ext} is responsible for external forces. (Baharudin 2016, p.27.)

These values can be calculated in the next way:

$$\delta W_{iner} = \delta \mathbf{q} \cdot (\mathbf{M}\ddot{\mathbf{q}} - \mathbf{Q}_v) \tag{12}$$

$$\delta W_{ext} = \delta \mathbf{q} \cdot \mathbf{Q}_e \tag{13}$$

Where, $\mathbf{M}$ represents the mass matrix, $\mathbf{Q}_v$ is the quadratic velocity vector, $\mathbf{Q}_e$ is the vector of external generalized forces. (Baharudin 2016, p.27.)

Therefore, the equation of motion can be expressed as:

$$\delta \mathbf{q} \cdot (\mathbf{M}\ddot{\mathbf{q}} - \mathbf{Q}_v - \mathbf{Q}_e) = 0 \tag{14}$$

2.2 Introduction to Artificial Intelligence, Machine Learning and Deep Learning

In the beginning, it is important to understand three basic concepts which refer to Artificial Intelligence. The basic definitions of artificial intelligence, machine learning and deep learning are introduced below.

2.2.1 Artificial Intelligence

Intelligence can be interpreted in several ways, it is known as the ability to learn how to act in previously unknown situations, as well as the ability to do the correct decisions according to some settings. As the example, a standard computer can be defined as an intelligent system that is capable to perform the actions according to predefined rules and settings. (Salvaris et al. 2018, p. 3.)

However, some typical operations performed by humans cannot be set by formal rules and settings for the machines to execute. It is impossible to make a code from the all knowledge and decision-making processes related to the information into a properly operating system that would react in the appropriate way. Comparing to the machines, humans are able to gain the knowledge and experience constantly that allow them to develop the decision-making base and increase the level of intelligence, moreover, to develop abstract thinking. (Salvaris et al. 2018, p. 3.)

AI itself is a field of study which aimed at combining possible computational performance of the machines and human ability to learn, sense and see the patterns. One of the target is to make intelligent machines, where their thought processes would be similar to humans, having the ability to simulate intelligence and make the decisions through the procedure in a similar way to human reasoning. This area of studies combines the approaches for operations and task performed in the past only by being intelligent to the aspiration to force machines to learn abstractedly and react on the feedback. (Salvaris et al. 2018, pp. 3-4.)

2.2.2 Machine Learning

Machine learning is known as a field of computer science where the system is trained by given data, this makes possible for the machines to learn the data and make the decisions.
Common ML tasks are dealing with regression, classification and clustering. AI can be identified as broader concept than Machine Learning, therefore, ML can interpreted as a subfield of AI. (Salvaris et al. 2018, p. 9.)

There are existing approaches how to use the data that represents the useful features or represents itself, for example: temperatures, stock prices, age, gender, country etc. The goal of the machine is to learn from this data and find the connections between the input set and the output features that are needed to predict. Engineers typically manually set the representations of the data, that process is called feature engineering, then the data is fed into the system to be learnt. Commonly, supervised machine learning is used, this means that the system is fed by the data which represents the ground truth against which the system is trained. (Salvaris et al. 2018, p. 9.)

Nowadays, Machine Learning is technique that is natural for many industries. It can be used for example in a predictive maintenance management, smart building management, supply management, customer relationship management, financial forecasting, churn prediction etc. In the field of mechanical engineering it is the tool for data analysis, design optimization, prediction design, quality assurance, preventive maintenance etc. (Salvaris et al. 2018, p. 12.)

2.2.3 Deep Learning

Deep learning (DL) is an area of Machine Learning, which goal is to develop machines which are able to learn. In general, DL is a batch of methods in the field of ML, where the distinctive feature is the use of NN, which working principle is similar to the operational principle of human brain. These days DL is widely used for text recognition problems, computer vision, pattern detection and other fields, moreover, it is getting popular in robotics, self-controlling cars etc. (Pattanayak 2017, p.1.)

ML is suitable for many traditional problems, however, there are many scenarios that do not have easily extractable features. DL can be classified as the next subfield of AI and ML, that gives a desired performance for the problems with a complex semantics: video, images, audio, text etc. In this method, a Deep Neural Network (DNN) with multiple layer is used with a huge amount of data. DNN sometimes can have the millions of parameters and need a big amount of data to conduct the training of the network. The aim of the network is to make a path from input to output. For example, it can be the map from the single pixels to the recognition of symbols, from audio samples to speech recognition etc. In such networks the input data is proceeded through the number of functions, the goal of the network is to find the optimal configuration of the weights in order to obtain the desired performance where the predicted outcome is close enough to the true labeled data. (Salvaris et al. 2018, p. 15.)

It is still necessary to preprocess the data before feeding the Neural Networks, but there is no need to set the features manually, because the network can be fed by raw input directly. DNN is able to work with the features automatically. Although, shaping and processing of the data is more simple, the structure of the network requires more effort while selection: number of neurons, number of layers, activation functions etc. (Salvaris et al. 2018, p. 15.)

For better understanding, the levels and hierarchy of AI are presented below in Figure 3.

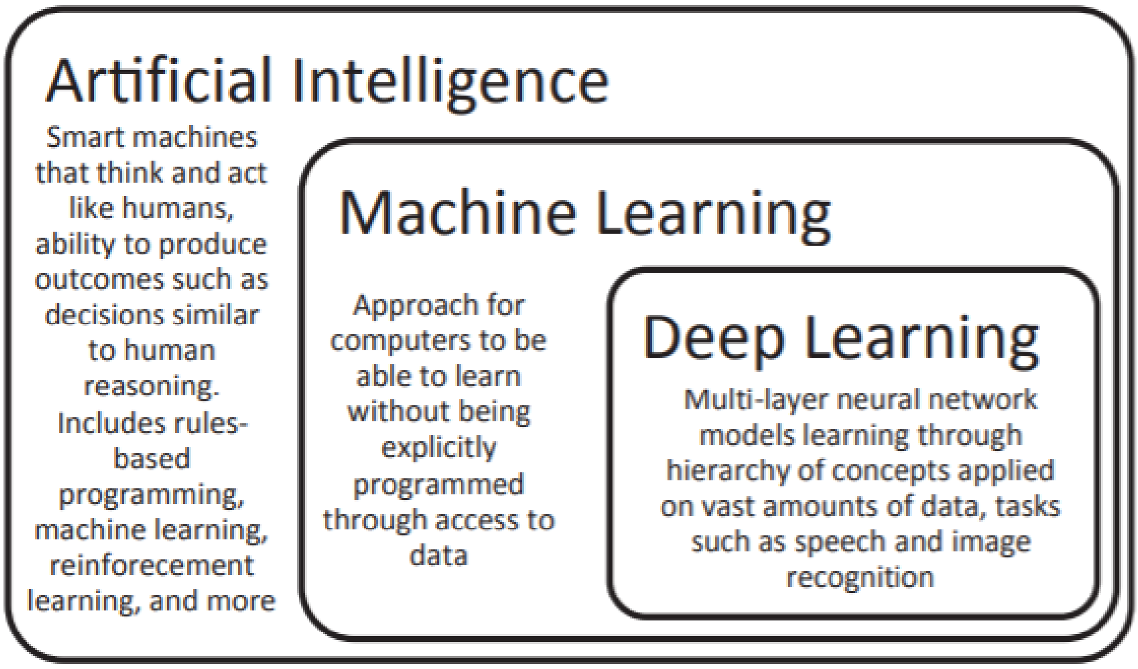

Artificial Intelligence
Smart machines that think and act like humans, ability to produce outcomes such as decisions similar to human reasoning. Includes rules-based programming, machine learning, reinforecement learning, and more
Machine Learning
Approach for computers to be able to learn without being explicitly programmed through access to data
Deep Learning
Multi-layer neural network models learning through hierarchy of concepts applied on vast amounts of data, tasks such as speech and image recognition

Figure 3. Artificial intelligence levels (Salvaris et al. 2018, p. 25.).

2.2.4 Supervised and unsupervised learning

In Machine Learning systems learn the patterns and try to mimic it in the certain approach that can be recognized either direct or indirect, in terms of ML they can be defined as supervised and unsupervised. (Trask 2019, p. 12.)

In supervised Machine Learning, every single component of training data is connected with the number of input features, commonly with an input feature vector and a corresponding vector. A system is configurated with the number of parameters that are aimed at a prediction of an output label using the input feature vector. The parameters of the system can be found by optimizing the function that is obtained while calculating the difference between the real labels and the predicted ones. (Pattanayak 2017, p. 56.)

In unsupervised Machine Learning, the process of learning is based on finding the patterns in the input data without labels and targets. An example of unsupervised learning is shown below in Figure 4. The input data consists of five words, after training the network it can be categorized into two groups. (Pattanayak 2017, p. 65.)

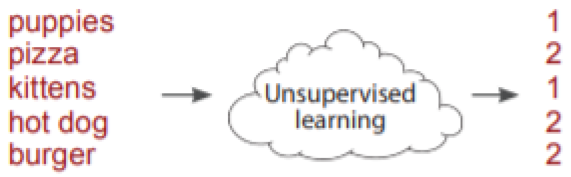


Figure 4. Unsupervised learning (Trask 2019, p. 13.).

2.3 Feedforward Neural Networks

Feedforward neural networks (FFNN) are such NNs where the connections between neurons are only possible from neurons in layer a to neurons in layer a + 1. The backward and connections inside the layers are not possible in this type of NNs. This fact defines the name of such networks. (Rosebrock 2017, p.127.) In FFNNs the data gets into the network at the input layer and goes through the network passing each layer until it reaches the output layer. (Michelucci 2018, p.83.) In this chapter the essential components of feedforward neural networks are covered.

2.3.1 Neuron

DNNs are dealing with big and complex networks which are formed from a big number of elementary units which are performing specific calculation. In a simple way, neuron takes a predefined number of input and processes it to output. These units are known as neurons. Artificial neurons can be set in specific way, changing the computational way, connection between each other, use of input data. These settings define the architecture of NN. (Michelucci 2018, pp. 31-32.)

The graphical representation of a basic neuron can be seen in Figure 5. This can be interpreted in the next way:

- The values from x_1 to x_{n_x} can be defined as inputs.
- The values from w_1 to w_{n_x} can be defined as weights. Before input is going through the central node, it is multiplied by the current weight. Each weight corresponds to the certain input. The measure of weight defines the relative importance of input.
- In the central node several calculations take place. In the beginning the inputs are summed up, then the bias b is added to this sum. Finally, an activation function is applied. (Michelucci 2018, p. 34.)

The output is following formula:

$$\hat{y} = f(z) = f(w_1 x_1 + \cdots + w_{n_x} x_{n_x} + b) \tag{15}$$

Where, z is a neuron, f is a neuron activation function.

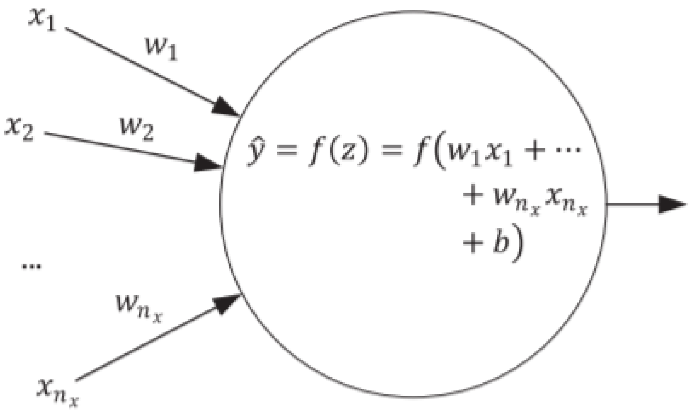

x_1
w_1
x_2
w_2
$\hat{y} = f(z) = f(w_1 x_1 + \cdots + w_{n_x} x_{n_x} + b)$
...
w_{n_x}
x_{n_x}

Figure 5. Neuron representation (Michelucci 2018, p. 34.).

2.3.2 Layers

In multilayer NNs, artificial neurons are grouped into separated arrangements which are known as layers. Typical multilayer NN has the next structure:

- One input layer
- One or more hidden or inner layers
- One output layer

Figure 6 shows an example configuration of Neural Network. In this network the neurons from each layer are fully connected with the neurons in the next layers. Typically, each neuron in the single layer use the same activation function. In the input layer, the input data is presented in the form of the raw vector. Being the inputs for one layer, at the same time it is outputs for the previous layer neurons of the other layers is the output (activation) of the previous layer's neurons. While the data is passed through the network it is transformed by the weights and activation functions which are presented in the given configuration of the NN. (Patterson and Gibson 2017, p.54.)

Input layer is made to feed the input data into the NN. Commonly, the number of neurons is equal to the number of input features in the certain problem. Input layer is usually fully connected with the first hidden layer. (Patterson and Gibson 2017, p.55.)

Hidden layer is the next layer after the input layer. The number of hidden layers varies depending on the problem under consideration. The weight rate on the connections between these layers show the way how the NNs encode the learned data which was extracted from the training set. Hidden layers are an essential part on the way to modelling the nonlinear function that distinguishes them from single- layer networks. (Patterson and Gibson 2017, p.55.)

Output layer is aimed at making the prediction by processing the data from the input layer through the hidden layers. Depending on the problem and configuration of the NN, the final output can exist in the form of a real value result corresponding to the regression problems or a set of probabilities that are related to the classification problems. This is reflected by the activation function utilized in the output layer. (Patterson and Gibson 2017, p.55.)

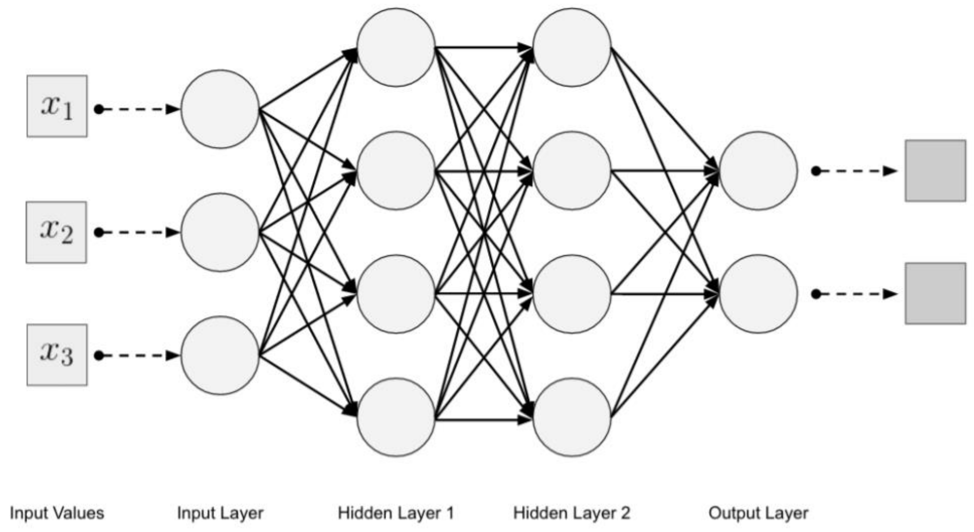


Figure 6. Multilayer neural network topology (Patterson and Gibson 2017, p.55.).

2.3.3 Activation function

In the NN activation function it is needed to distribute the output of the nodes in in layer ahead to the following layer, up to the last layer. Activation function is a scalar-to-scalar function. The aim of activation function is to give nonlinearity to the hidden layers of the NN. There is a number of different activation functions that are used in Machine Learning, their application is mostly depending on the type of considered problem and required output. The most common activation functions are listed below. (Patterson and Gibson 2017, pp. 65-66.)

Linear

Linear transformation can be interpreted as the identity function, in this function the dependency of dependent and independent variables is characterized as a proportional and has a direct relationship. Basically, the signal passed through stays unchanged. The graph of such function is shown in Figure 7. (Patterson and Gibson 2017, p.66.)

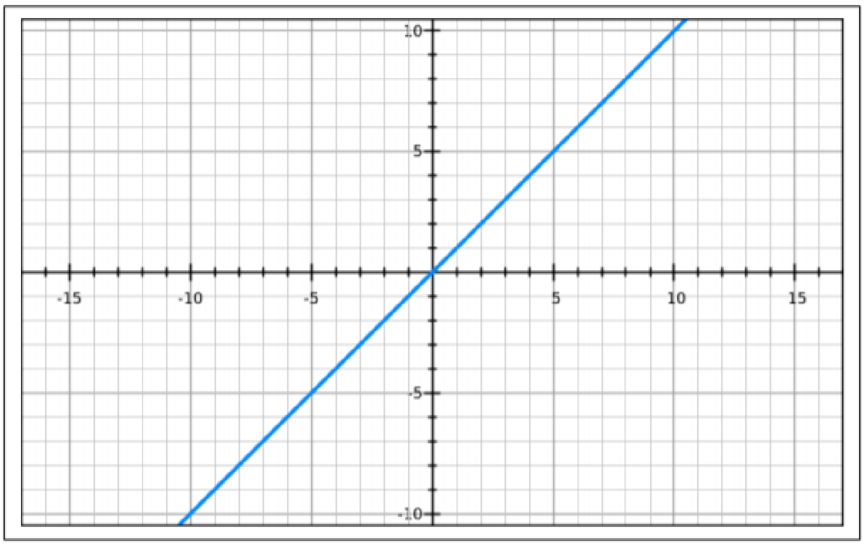


Figure 7. Linear activation function (Patterson and Gibson 2017, p.66.).

ReLu

Nowadays, ReLu is one of the most popular activation functions. The main advantage is that Relu accepts only positive values and blocks negative values, this gives the possibility to increase the learning speed of NN. It means that when the input is less than zero, output is also zero, but while input is increasing the output has a proportional relationship with the dependent variable. The advantage of this function that it is not subjected to vanishing gradient problems because it never saturates. The equation and graph are shown below. (Patterson and Gibson 2017, p.69.)

$$f(x) = max(0, x) \tag{16}$$

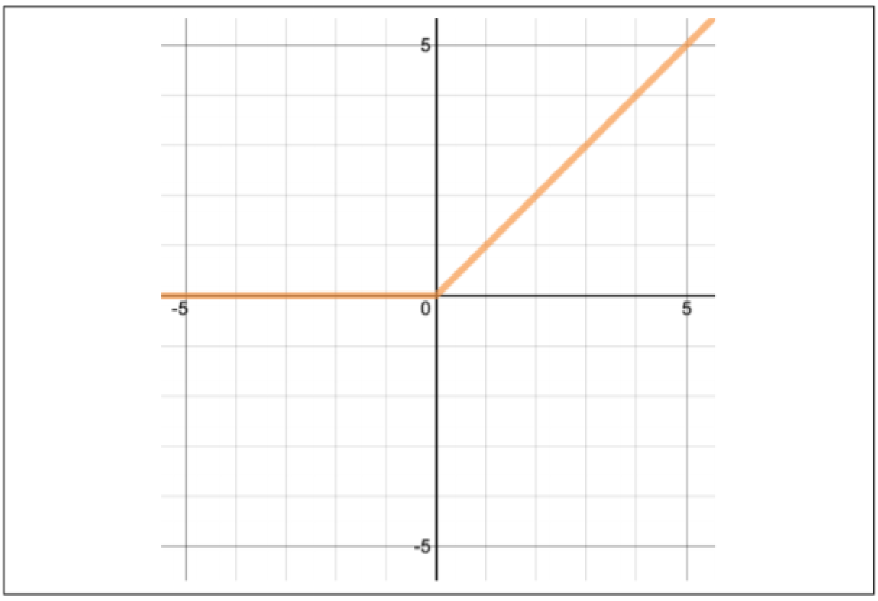


Figure 8. ReLu curve (Patterson and Gibson 2017, p.69.).

Softplus

Softplus can be identified as the ‘soft’ setting of ReLu activation function. The graph of the function is almost the same as ReLu. At the same time there is no zero derivative in the all range where the function is. The equation and graph are illustrated below. (Patterson and Gibson 2017, p.70.)

$$f(x) = ln\,[1 + exp(x)] \tag{17}$$

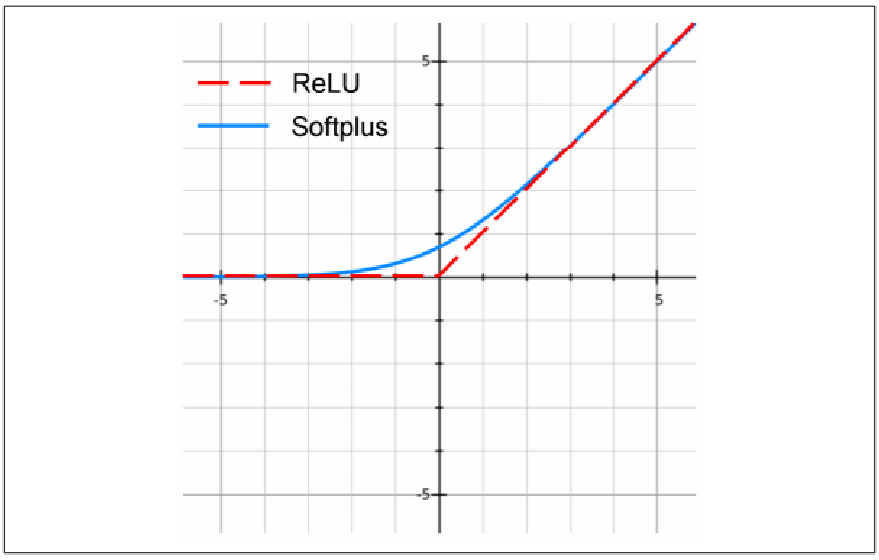


Figure 9. Softplus activation function (Patterson and Gibson 2017, p.70.).

Tanh

Tanh is a hyperbolic trigonometric activation function which scales the values in the range between -1 and 1. The good point of this function is the ability to proceed easily negative numbers. (Patterson and Gibson 2017, p.67.)

The function and graph are shown below.

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{18}$$

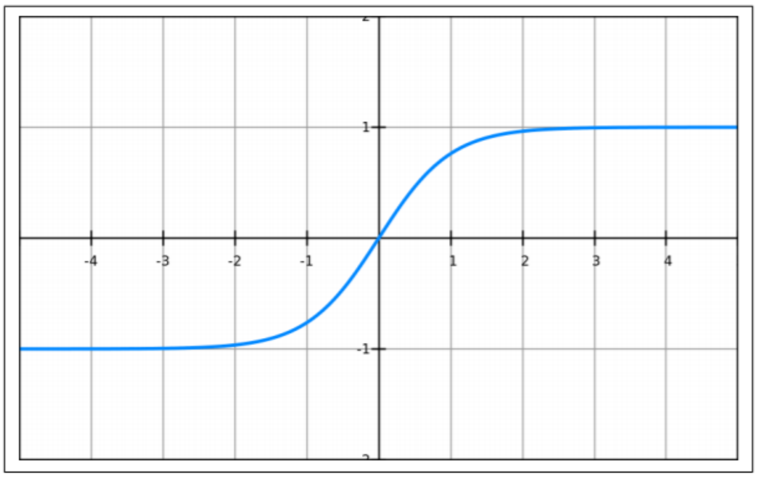


Figure 10. Tanh function (Patterson and Gibson 2017, p.68.).

Sigmoid

The sigmoid function is a non-linear activation function which scales the values in the range between 0 and 1. Sigmoids are capable to decrease extreme value in the data but keeping them without removing. It converts independent variable into basic probability. The graph of the function and its formula are shown below. (Patterson and Gibson 2017, pp. 66-67.)

$$f(x) = \frac{1}{1 + e^{-(x)}} \tag{19}$$

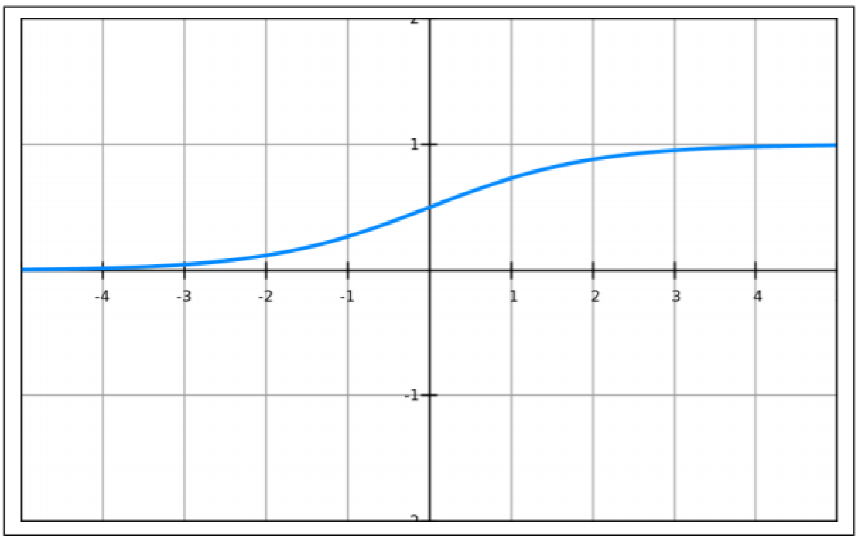


Figure 11. Sigmoid curve (Patterson and Gibson 2017, p.67.).

2.3.4 Loss function

Loss functions are aimed at quantifying how close a considered Neural Network is to the system that it is training. In this case the error value based on the difference between predicted and real values is calculated. Then the errors over the whole dataset are aggregated and after averaging the total error a single value shows how NN is close to the system under consideration. (Patterson and Gibson 2017, p. 71.)

There is a certain number of loss functions which are mainly used in regression problems.

Mean Squared Error

Mean Squared Error (MSE) shows the average squared difference between actual and predicted values. Due to squaring, when the difference is bigger squared value is also bigger, that simplifies the process of penalizing the model with higher difference. (Patterson and Gibson 2017, p.73.)

This loss is represented by the next mathematical formulation:

$$MSE = \sum_{i=1}^{n} \frac{\left|y_i - y_i^p\right|^2}{n} \tag{20}$$

Where, y_i is actual value, y_i^p is a predicted value, n number of predictions.

Mean Absolute Error

Mean Absolute Error (MAE) is averaging the absolute error through the whole dataset. (Patterson and Gibson 2017, p.74.)

The mathematical formulation is the next:

$$MAE = \sum_{i=1}^{n} \frac{\left|y_i - y_i^p\right|}{n} \tag{21}$$

There are two more loss function which are not so popular, but still can be found in the configuration of some NNs. These functions are: Mean squared log error (MSLE) and Mean Absolute percentage error loss (MAPE). (Patterson and Gibson 2017, p.74.)

As can be seen, there is a number of different loss functions. Their use is mainly dependent on the application and there is no possibility to choose the single function that would be suitable for all the scenarios. The MSE is the most popular choice for regression problems. At the same time, MSLE and MAPE should be considered while dealing with the problems where the outputs are significantly varying in range. In those problems where MAE and

MSE would penalize one of the output variables more significantly, MAPE and MSLE are able to avoid the discrimination caused by the different ranges. (Patterson and Gibson 2017, p.74.)

2.3.5 Optimizers

After defining the loss, it is possible to evaluate the current setting of the network, to check the performance of weights and to define how and in which direction it should be adjusted. This procedure can be implemented by optimizer function. While training the model the goal is to find the best combination of parameter vector of system under consideration. It follows the desire to decrease the loss function according to the settings of the prediction function. There is a number of techniques that are commonly used to find the best set of parameters. (Patterson and Gibson 2017, p. 27.)

Stochastic Gradient Descent

Stochastic Gradient Descent is an optimizer which working principle is based on calculating the gradient of the loss function and then updating the weights and biases for each training example separately. It goes over the whole dataset the number of times until it finds the appropriate parameters that would work satisfactorily for the whole training dataset. Frequent updates make possible to follow the learning process better, it the advantage of the method, at the same time, on the large datasets SGD can slow down the training process, because the method is computationally intensive. (Michelucci 2018, p. 115.)

ADAM

Adam is known as Adaptive Moment Estimation Algorithm is a relatively new optimization algorithm that was introduced in 2015. The principle of the method is dealing with the estimation of moments and then it used to optimize the parameters. It is calculating an exponential weighted moving average of the gradient, after the gradient is squared. Simple implementation, computational efficiency, efficient memory use, feasibility for big datasets and adaptiveness for noisy gradients are the advantages of the algorithm. (Battini 2018)

2.4 Recurrent Neural Networks

Recurrent Neural Networks belongs to the type of Feed-Forward Neural Networks. The advantage of such NNs over the other FFNN is the capability to send the data over time-steps. RNNs use each vector from an array of input vectors and replicate them one per unit of time. This enables the possibility to keep state when modeling the input vectors across the sequence of input vectors. Modeling time dependent data is the main advantage of RNN. (Patterson and Gibson 2017, p.143.)

The operational principle of such networks is shown in Figure 12.

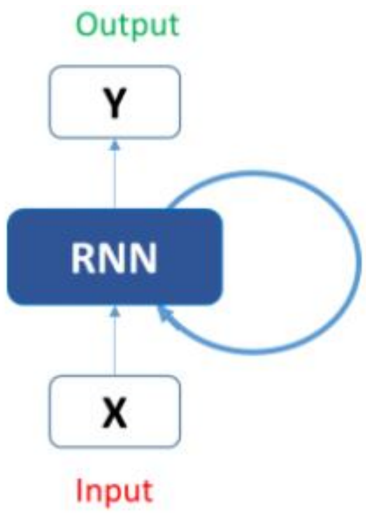


Figure 12. RNN unit (Shaikh, 2019).

As it can be seen, the connections in such networks allows to make visible not only current state fed into network, but also use the previous states. It works as a network with a feedback loop, where output acts as an input for the next network. Comparing to the other types of NNs, RNN are aimed at evaluating and analyzing the data from the past in order to make the prediction for the future value, while the traditional NN makes every step from the scratch. (NVIDIA Developer 2019)

RNN are widely used for analyzing the sequence of data, which is related to classification problems, regression estimation. Capability to use the past data is making such networks attractive to utilize in the problems which are related time series data. (NVIDIA Developer 2019)

Nevertheless, due to loop principle of such networks, it causes large transformation of NN model weights, as the result the network accumulates error of gradients while updating, this leads to lose of stability of the network. When gradients are multiplied continuously through the layers with the values greater than 1, it leads to explosion, at the same time when values are less than 1, vanishing takes place. (NVIDIA Developer 2019)

2.4.1 LSTM

To overcome the problem described in the previous chapter Long Short Term Memory (LSTM) RNN are commonly used. The configuration of LSTM and the components are shown below in Figure 13 and Table 1.

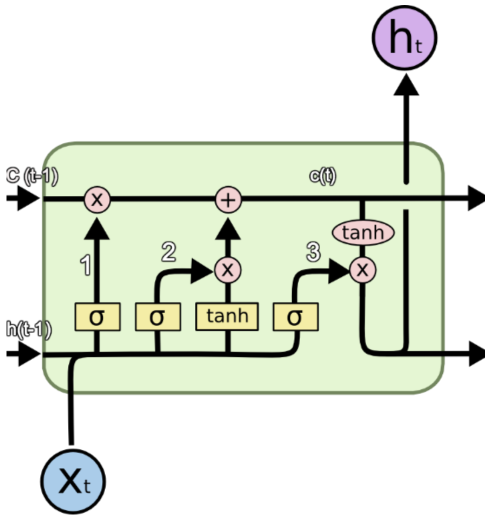


Figure 13. LSTM unit (Sinha 2018).

Table 1. LSTM components (Sinha 2018).

Symbol	Description
X	Information scale
+	Information add
σ	Sigmoid layer
Tanh	Tanh layer
h(t-1)	Output from previous LSTM component
c(t-1)	Memory from previous LSTM component
X(t)	Input
c(t)	Updated memory
h(t)	Updated output

Tanh layer is used to resolve vanishing gradient issue, because the second derivative of this function is resistant for a long time before getting zero. Sigmoid function is used for keeping or deleting the information. (Sinha 2018)

Sigmoid structure allows to keep or forget the information, depending on its importance. Then the information from the next input is stored in the cell state, sigmoid layer is evaluating whether this information must be updated or avoided. A Tanh layer builds a vector consisting of the range of values available from the new input. Then these layers are multiplied to change the cell state. After this, memory is updated. (Sinha 2018)

At the last stage the decision about output should be made. A sigmoid layer is responsible for making the decision about cell state, what is data is the output. After this, tanh layer is used to create a combination of possible values, which are multiplied by output of sigmoid gate. (Sinha 2018)

So, it can be seen that the model is learning from the dependency which comes from the past, meaning that the output is based on the previous data, at the same time network is able to process the data and manage it. (Sinha 2018)

Summing up, LSTM shows the better performance comparing to the other types of NNs when it is needed learn the patterns from long term dependencies. LSTM’s feature to forget, remember and update the data brings it to the other level of performance comparing to the ordinary RNNs. (Sinha 2018)

2.5 Neural Networks in Keras

This graph represents the use of Keras for Neural Networks.

2.5.1 Introduction to Keras

Keras is a package which is used for building NNs in Python. Using Keras module for Python, it is possible to build a Neural Network which could be trained by input data and later used for value prediction. It is known as a high-level Application Programming Interface (API), which has user friendly interface, easy extendable environment and modularity. Keras provides an access to the NN’s standard building components: layers, activation functions, metrics, optimizers. Use of these blocks allows to build NN, which are able to work with dataset, learn from it and make the predictions. (Brownlee 2016)

2.5.1 Data representation

The data in Keras stored using tensors. Tensors itself can be defined as the containers for data, nowadays it the basic data structure which is being used by the systems of ML. Tensors are just the generalization matrices for a capricious number of dimensions. Dimensions in terms of tensors are frequently referred to axis. (Chollet n.d., p. 30.)

Scalars can be defined as 0D tensors, scalar is real number and also a component of a field utilized to define a vector space. An array of vectors is defined as 2D tensor or matrix, which has two axes. When matrices are combined into a new array, it is called 3D tensor, which is represented as a cubic set of number. While processing DL, it is mostly dealing with 4D and even 5D tensors when it is connected with video processing. (Chollet n.d., pp.30-31.)

2.5.2 Neural Network building

Firstly, the building of the NN in Python using Keras starts with the importing of needed packages which will utilized. This can include packages that are needed for reading and processing data from text file. Moreover, it includes packages needed for math operations, as well as components of NN taken from Keras that will be used for building a model. (Wei En 2019)

At the second stage it is necessary to prepare the data, scale it if needed, exclude extra data and split it for input features and output features that are going to be predicted. The data should be also split for training, validation and test sets. (Wei En 2019)

Training dataset is used for fitting the model. Validation dataset is made for unbiased evaluation of the performance of the model at current state on the training data at the hyperparameters tuning stage. Finally, test dataset is used for conduction of unbiased evaluation of the performance of model’s configuration fit on the training dataset. (Brownlee 2017)

At the next stage, the architecture of the model should be configurated. This includes number of layers, especially hidden layers, then it is necessary to take into consideration number of neurons in each layer, as well as activation function for each layer. (Wei En 2019)

Then, it is necessary to specify algorithm which will be used for optimization, loss function and metric, this is known as a compiling the model. The next step is to fit chosen parameters to the data. At this stage it is also important to choose the size batch and number of epochs. Batch size refers to the number of data samples used per gradient update while training. Number of epochs shows the number of iterations over the whole data to train for. (Wei En 2019)

Next, the training is performed. At this stage the set of best hyperparameters is being selected. This done by analyzing metrics that are outputs at this stage. If the result is not satisfying, the architecture of the model is subjected to changes. It continues until the desirable result is reached. (Wei En 2019)

Finally, the performance of the model can be evaluated on the test set and saved for the future use. (Wei En 2019)

The operational principle of NN is shown in Figure 14. The network has the number of layers which are connected, it turns the input data into predictions. Then loss function is used to compare the predictions to the desired values. Then the optimizer takes the loss value in order to modify the weights. Then the input data will be proceeded through the network with the updated weights again. The process continues as long as it was set while designing the network. (Chollet n.d, p.14.)

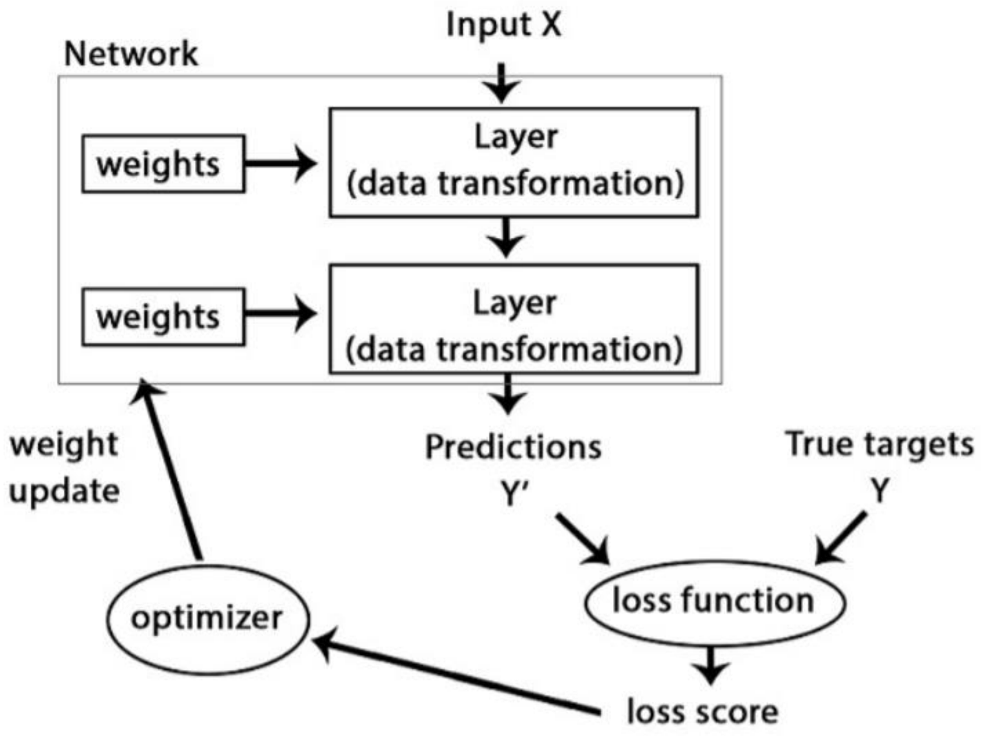


Figure 14. Neural Network process flow (Chollet n.d, p.14.).

3 RESEARCH METHODS

This chapter is about a research implementation. The process consists of several parts. Firstly, the implementation of the technology for a system of two nonlinear functions is introduced. Secondly, the technology is applied to a double pendulum.

3.1 Software

Software which was used during the practice is shown below in Table 2. The most of work was conducted in Python. It was used for a building of the NNs, which was trained and tested by the data obtained from the hdf5 dataset. Matlab was used to create a data set, where all the configurations of function *f(1)* and *f(2)* were saved. Matlab was used to derive the equations of motion of double pendulum.

Table 2.Software

Sofware	**Version**
Python	3.6
Matlab R2018a	9.4.0

3.2 System I

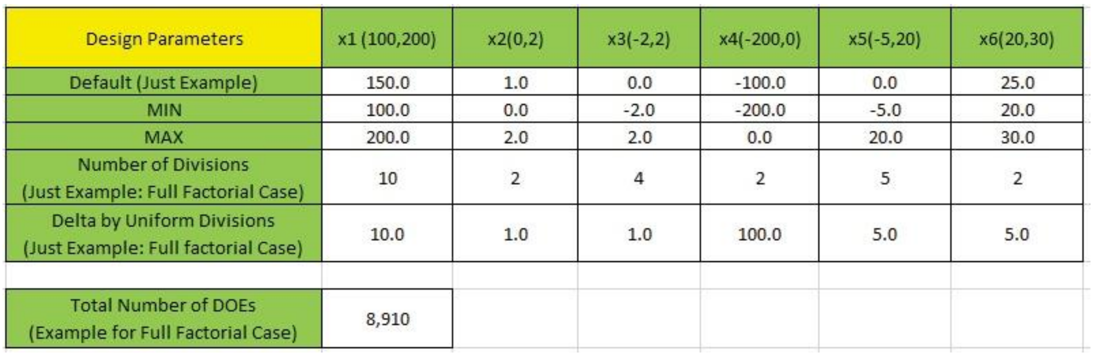
At the first stage of the project, the system with 6 variables and 2 configurations was chosen. The first function is defined using equation 22:

$$f(1) = x1 \cdot x2 \cdot \ cos(x2) \ \cdot sin(10 \cdot x3) + \frac{x2 \cdot x2 \cdot x4}{\sqrt{|x4+1| \cdot \ x5 \cdot \ \sqrt{x6}}} \tag{22}$$

The second function is defined by equation 23:

$$f(2) = x3 \cdot x3 \cdot (x1 \cdot\ x2 \cdot cos(x2) \cdot sin(10 \cdot x3) - \frac{\frac{x2 \cdot x2 \cdot x4}{\sqrt{|x4+1| \cdot x5 \cdot \sqrt{x6})}}}{\sqrt{x6}} \qquad (23)$$

The parameters for variables are shown in Figure 15:



Design Parameters	x1 (100,200)	x2(0,2)	x3(-2,2)	x4(-200,0)	x5(-5,20)	x6(20,30)
Default (Just Example)	150.0	1.0	0.0	-100.0	0.0	25.0
MIN	100.0	0.0	-2.0	-200.0	-5.0	20.0
MAX	200.0	2.0	2.0	0.0	20.0	30.0
Number of Divisions (Just Example: Full Factorial Case)	10	2	4	2	5	2
Delta by Uniform Divisions (Just Example: Full factorial Case)	10.0	1.0	1.0	100.0	5.0	5.0
Total Number of DOEs (Example for Full Factorial Case)	8,910					

Figure 15. Function variables settings

3.2.1 Keras model topology

The model consists of four layers: an input layer, two inner hidden layers and an output layer.

Input layer has the dimension of 6, because the total number of input variables is 6. The output from input layer is the input for the first hidden layer. This layer has 128 neurons, each neuron of this layer is connected with each neuron from input layer, as well as the connection with each neuron from the next hidden layer. The output of first hidden layer has the dimension of 128, they serve as inputs for the second hidden layer, each perceptron is fully connected with each perceptron from adjacent layers. The second hidden layer that has the dimension of 64 is connected to output layer which has two outputs, corresponding to the number of functions.

To make it clear, the topology of the model is shown in Figure 16.

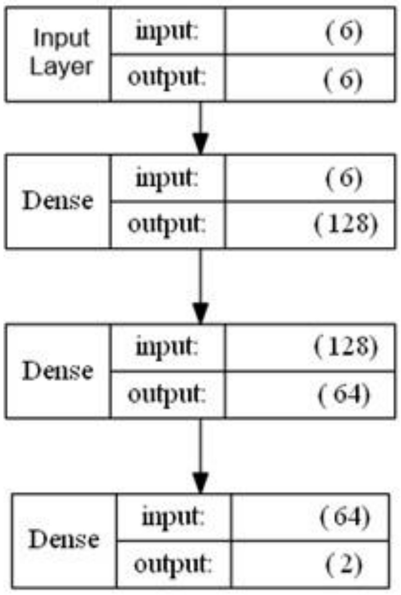
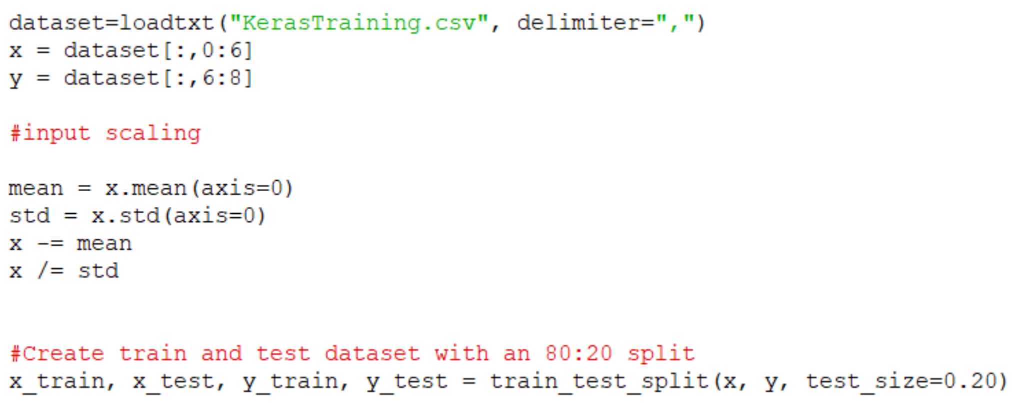


Figure 16. Keras model topology

3.2.2 NN model implementation

In this task NN is implemented in order to predict function value with given $x_1 - x_6$ variables values.

The dataset has 8 910 datapoints, which are split between training data and test data as 80:20, meaning that 7 128 datapoints are used for training of the Neural Network and 1 782 are test samples. Some of training data is used for validation, which is essential part for analyzing possible overfitting, this will be introduced later. As it can be seen, input variables have different ranges. To make it easier for learning, the input data should be normalized. To center the variables around 0 and to get a unit standard deviation, the mean of variable is subtracted and then divided by the standard deviation. These first steps are shown in Figure 17.

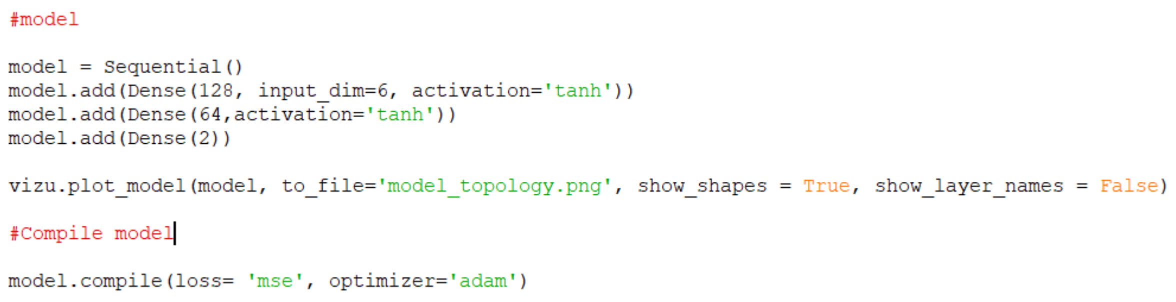


```
dataset=loadtxt("KerasTraining.csv", delimiter=",")
x = dataset[:,0:6]
y = dataset[:,6:8]

#input scaling

mean = x.mean(axis=0)
std = x.std(axis=0)
x -= mean
x /= std


#Create train and test dataset with an 80:20 split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20)
```

Figure 17. NN building

The next step is to build a network. As it was mentioned before, the network has two hidden layers, 128 neurons in the first layer and 64 in the second. In the output layer, network has two neurons and there is linear activation function which is natural for scalar regression, where the goal is to predict a continuous value, meaning that the network is able to predict the output in any range without constraining.

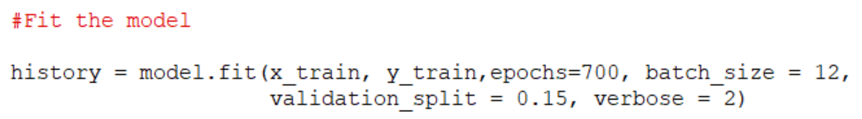
While compiling the model, MSE loss function and Adam optimizer are used, which is typical for regression problems. The implementation of this part is shown in Figure 18.



```
#model

model = Sequential()
model.add(Dense(128, input_dim=6, activation='tanh'))
model.add(Dense(64,activation='tanh'))
model.add(Dense(2))

vizu.plot_model(model, to_file='model_topology.png', show_shapes = True, show_layer_names = False)

#Compile model

model.compile(loss= 'mse', optimizer='adam')
```

Figure 18. NN Configuration

When the model is configurated, the next stage is to train it using data. In order to utilize this process, a piece of data should be used for validation. The idea is to check the overall performance of the model. First, the model is using training set to learn the patterns, after this the initially unseen validation set is used for predictions and metrics evaluation. To conduct a training process in Keras fit function is used. Training implementation lines from Python are shown in Figure 19.



```
#Fit the model

history = model.fit(x_train, y_train,epochs=700, batch_size = 12,
                    validation_split = 0.15, verbose = 2)
```

Figure 19. Training of the model

As it can be seen, the model is trained by *x_train* and *y_train*, which were defined in the beginning. The number of epochs is 700 and batch size is equal to 12, meaning that 12 units of training data are processed at a time, then the weights are changed and the next set of training data is processed. There are 700 rounds of training. Finally, the performance is validated by validation set, 15% of training data is used for this purpose.

3.3 Double Pendulum

This part contains the information about the double pendulum, configuration of the system and parameters, also, derivation of the equations of motion and designing of NN are covered.

3.3.1 System configuration

In the next stage of the project double pendulum is investigated. Double pendulum is a dynamical system, which consists of two connected links. Also, a torsional spring is located between links. The behavior of double pendulum is sensitive to the initial parameters and can be represented by the system of ordinary differential equations. The system is shown in Figure 20.

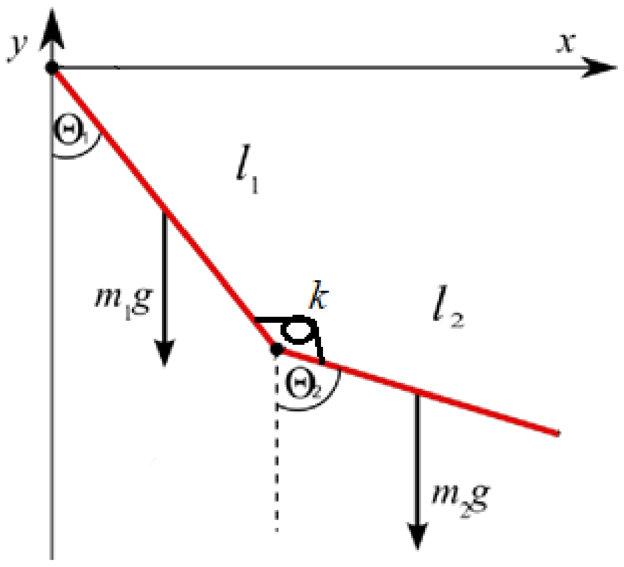


Figure 20. Double pendulum

3.3.2 System parameters

The parameters of double pendulum can be observed in Table 3.

Table 3. Double Pendulum System parameters

Symbol	Description	Value	Unit
m_1	Mass of link I	2.689	*kg*
m_2	Mass of link I	0.537	*kg*
l_1	Length of link I	0.4	*m*
l_2	Length of link II	0.6	*m*
I_1	Moment of inertia of link I	4.202E-02	*kgm*^{2}
I_2	Moment of inertia of link II	1.674E-02	*kgm*^{2}
g	Gravity constant	9.807	*m/s*^{2}
k	Spring coefficient	18/pi	*N/rad*

The initial values of the system are shown in Table 4.

Table 4.Initial values

Parameter	α_1	α_2	$\frac{d\alpha_1}{dt}$	$\frac{d\alpha_2}{dt}$
Initial value	pi/2	pi/2	0	0

3.3.3 Equation of motion of double pendulum

The equations of motion of double pendulum can be derived using Lagrangian Mechanics. The positions of the masses are given by:

$$x_{1p} = \frac{l_1}{2}\sin(\alpha_1) \tag{24}$$

$$y_{1p} = -\frac{l_1}{2}\cos(\alpha_1) \tag{25}$$

$$x_{2p} = l_1\sin(\alpha_1) + \frac{l_2}{2}\sin(\alpha_2) \tag{26}$$

$$y_{2p} = -l_1\cos(\alpha_1) - \frac{l_2}{2}\cos(\alpha_2) \tag{27}$$

Where l_1 and l_2 are lengths of the links of the double pendulum, α_1 and α_2 are angles. From these expressions the velocities of the masses can be derived:

$$\dot{x}_{1p} = \dot{\alpha}_1\frac{l_1}{2}\cos(\alpha_1) \tag{28}$$

$$\dot{y}_{1p} = \dot{\alpha}_1\frac{l_1}{2}\sin(\alpha_1) \tag{29}$$

$$\dot{x}_{2p} = \dot{\alpha}_1 l_1\cos(\alpha_1) + \dot{\alpha}_2\frac{l_2}{2}\cos(\alpha_2) \tag{30}$$

$$\dot{y}_{2p} = \dot{\alpha}_1 l_1\sin(\alpha_1) + \dot{\alpha}_2\frac{l_2}{2}\sin(\alpha_2) \tag{31}$$

Potential and Kinetic Energies of the system can be calculated using the next equations:

$$V = m_1 g y_{1p} + m_2 g y_{2p} + \frac{k(\alpha_2 - \alpha_1)^2}{2} \tag{32}$$

$$T = \frac{m_1}{2}\left(\dot{x}_{1p}^2 + \dot{y}_{1p}^2\right) + \frac{m_2}{2}\left(\dot{x}_{2p}^2 + \dot{y}_{2p}^2\right) + \frac{I_1\dot{\alpha}_1^2}{2} + \frac{I_2\dot{\alpha}_2^2}{2} \tag{33}$$

The Lagrangian of system is expressed as:

$$L = T - V \tag{34}$$

Dynamic equation of the system can be obtained by using Lagrangian mechanics, hence the equation of motion has the next form:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_j}\right) - \frac{\partial L}{\partial q_j} = 0 \tag{35}$$

Where, q_j refers to generalized coordinates, $\dot{q}_j$ is generalized velocity.

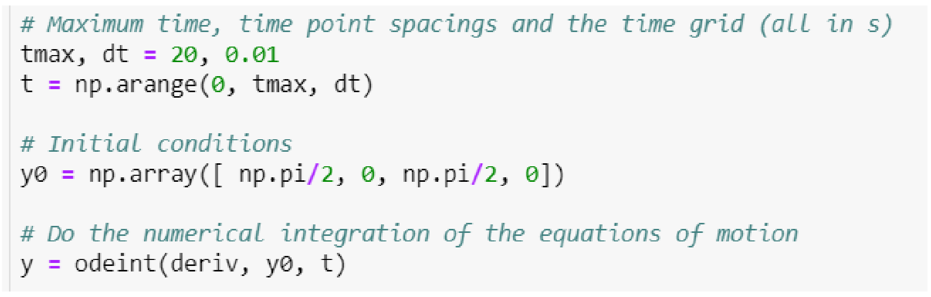
The components of the equation can be obtained manually and or computed by software. For this example, Matlab was chosen for deriving the equations of motion. The whole script can be found in the Appendix 1.

3.3.4 Building of LSTM Neural Network in Keras

As it was mentioned in Chapter 2, LSTM are good for time-series problems, moreover, such NNs are able to remember, forget and update the data. This fact argues the use of LSTM in the particular case.

After converting the equations into the system of first-order differential equations, the equations obtained from the previous step can be used in Python.

First, it is necessary to do the numerical integration of the equations of motion obtained from Matlab, using the code shown below in Figure 21.



```python
# Maximum time, time point spacings and the time grid (all in s)
tmax, dt = 20, 0.01
t = np.arange(0, tmax, dt)

# Initial conditions
y0 = np.array([ np.pi/2, 0, np.pi/2, 0])

# Do the numerical integration of the equations of motion
y = odeint(deriv, y0, t)
```

Figure 21. Numerical Integration of equations of motion

Then, the NN can be configurated in order to train using the data obtained from the equations of motions. It was decided to use 200 previous steps to predict the next step, and data was split for train and test as 80:20.

Next step is to do the design of NN. There are 4 layers overall, input layer has the dimension of 2, because in this example the aim is to work with the angles α_1 and α_2. The output from input layer is the input for the first hidden layer. There are 2 hidden layers, in the output layer there are two outputs. Bidirectional LSTM means that input runs in two directions, both from past to future and from future to past. There were tested several configurations with the different number of neurons in inner layers, also, the number of epochs was subjected to changes. The results of those different settings are presented in Chapter 4.

Then the model is trained. The number of epochs was subjected to change while testing various configurations, batch size of 8 was chosen. To evaluate the performance the validation set is used and set as 10% from training data.

4 RESULTS AND ANALYSIS

This chapter is used to present the results of the practical stage of the research and to analyze the obtained results. The practical part of the research includes two cases where the AI was utilized to investigate the feasibility of the technology inside the field of multibody dynamics. First part describes the results for system consisting of two nonlinear equations and the second part is used to represent the outputs of the second case which was dealing with the system of double pendulum with the links connected by torsional spring.

As it was stated before, the goal of the research was to discover the possibility to utilize the power of AI in the field of multibody dynamics especially for the heavy machinery. Moreover, it was important to evaluate the performance of the obtained systems, to consider time consuming aspects and analyze the designing features and nuances.

4.1 System of nonlinear equations

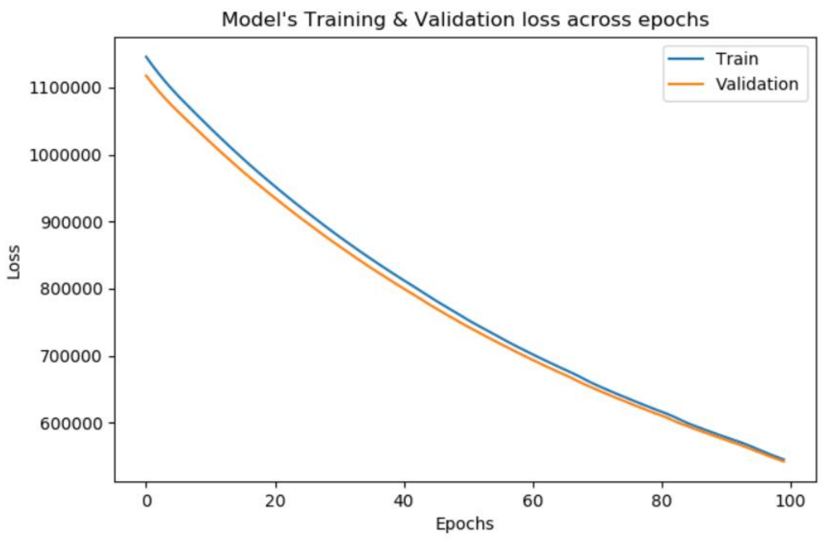
For the system of two nonlinear equations the NNs were implemented and tested in several configurations and with different settings in order to evaluate the performance of the NN and find the connection between the behavior and network design. These different configurations are presented and analyzed below.

4.1.1 Case I

In the first experiment the Feed Forward NN with one hidden layer with 16 neurons, trained for 100 epochs with batch size of 12 was set.

As it can be seen from Figure 22, the network has a poor performance, the losses are significant and the lines are aimed vertically down, meaning that the chosen configuration is not suitable and number of hidden layers, epochs should be modified.

Training time 00:38,29



```
Input data range min -6333.306986, max 4650.135995
Metrics loss value 560151.14 sqrt(value) 748.43 sv/range 6.81%
Losses train 5.454e+05 val 5.425e+05 test 5.602e+05
```

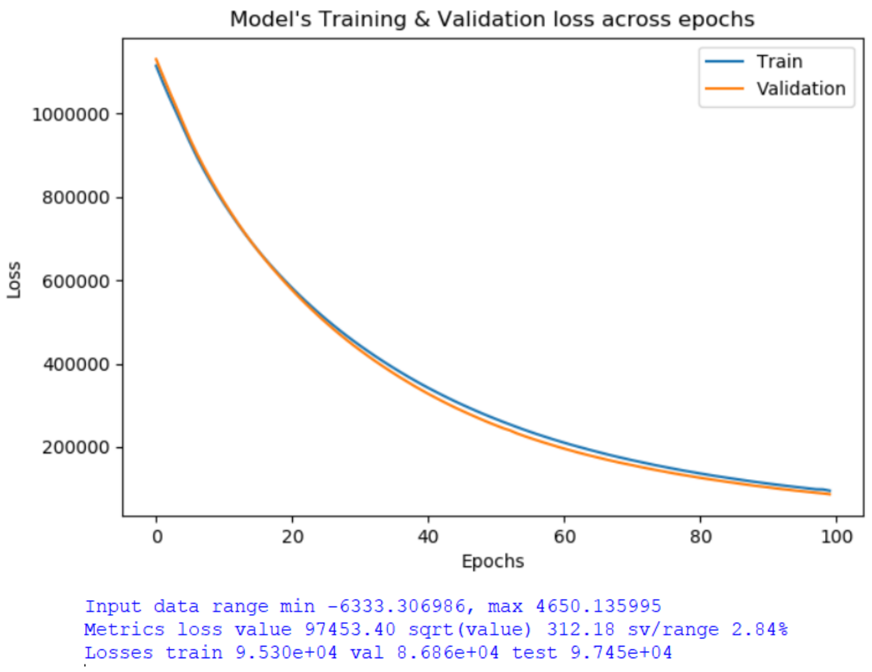
Figure 22. Performance of configuration I

4.1.2 Case II

In the second experiment the NN that has two hidden layers with 128 neurons in the first layer and 64 neurons in the second. The network was trained for 100 epochs with batch size of 12.

As can be observed from Figure 23, the performance of the system has improved comparing to the case I but still there is a space for the improvement. The losses are big, the values on the graph are still decreasing every epoch.

Training time 00:40,18



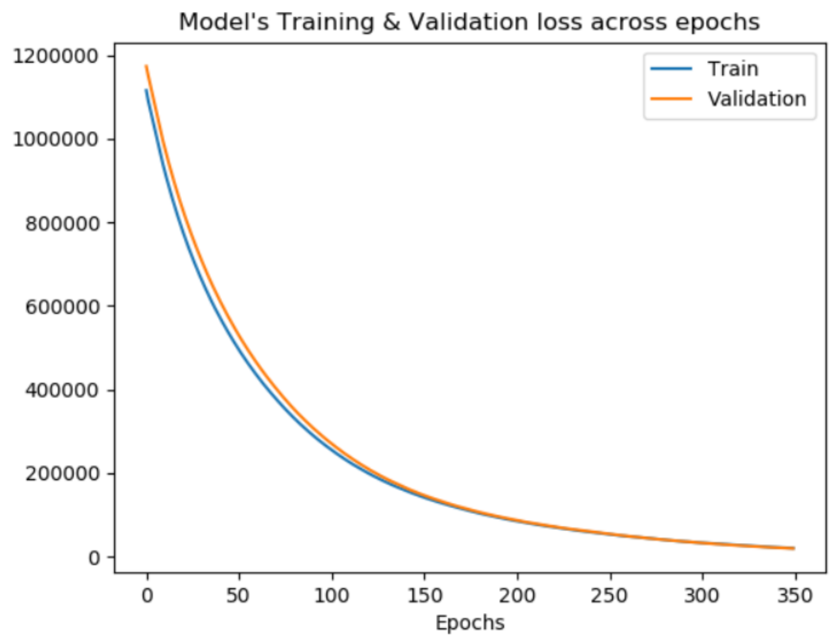
```
Input data range min -6333.306986, max 4650.135995
Metrics loss value 97453.40 sqrt(value) 312.18 sv/range 2.84%
Losses train 9.530e+04 val 8.686e+04 test 9.745e+04
```

Figure 23. Performance of configuration II

4.1.3 Case III

In the third experiment the configuration was modified in the way that the number of epochs was increased to 350. The rest parameters were the same.

It is obvious, that the performance of the system got better. The losses have decreased. It can be concluded that the number of epochs is important and should be set properly in order to get the desirable result. From Figure 24 can be seen that the number of epochs is still subjected to changes. In this case it took about 02:10,56 for the NN to proceed the data using the above-mentioned parameters.



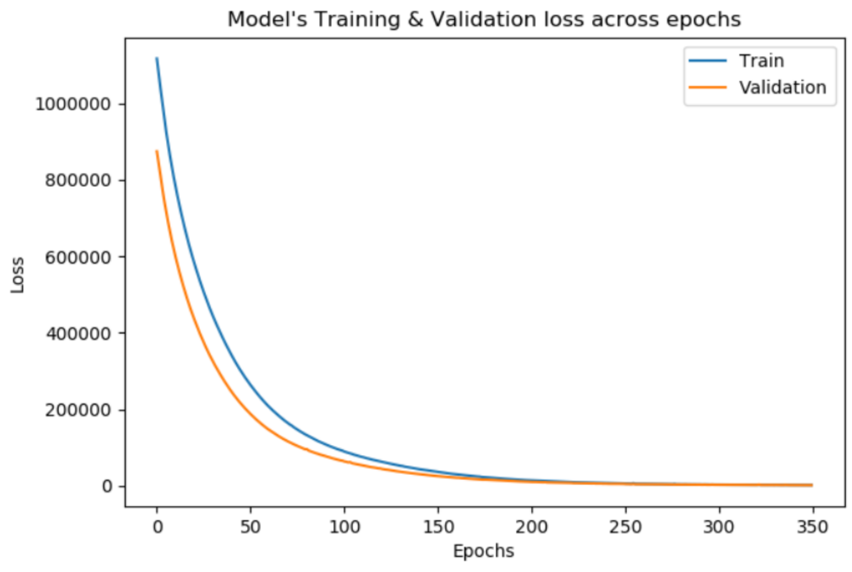
```
Input data range min -6333.306986, max 4650.135995
Metrics loss value 21260.78 sqrt(value) 145.81 sv/range 1.33%
Losses train 2.041e+04 val 1.989e+04 test 2.126e+04
```

Figure 24. Performance of configuration III

4.1.4 Case IV

In the next experiment it was decided to change the optimizer from Adam to Nadam. From the Figure 25, it can be concluded that the NN shows better performance, the losses have decreased and significant decrease in losses occurs at earlier stages using Nadam comparing to the results of previous experiment. This means that it is possible to decrease the time used for NN training by decreasing the number of epochs because use of Nadam enables faster training. At the same time execution time of the NN stayed approximately the same and equal to 02:10,38.

As the result, it was decided to continue working with the system using Nadam optimizer and the number of epochs is still needed to be increased.

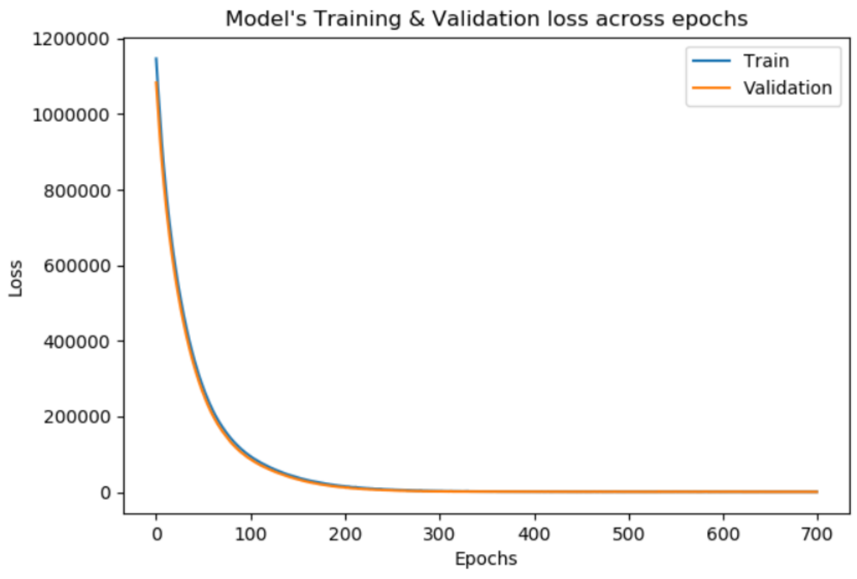


```
Input data range min -6333.306986, max 4650.135995
Metrics loss value 1486.95 sqrt(value) 38.56 sv/range 0.35%
Losses train 1.392e+03 val 1.199e+03 test 1.487e+03
```

Figure 25. Performance of configuration IV

4.1.5 Case V

In the last experiment the number of epochs was increased to 700 and the rest parameters were the same. It was done in order to decrease losses and get more accomplished NN.



```
Input data range min -6333.306986, max 4650.135995
Metrics loss value 483.18 sqrt(value) 21.98 sv/range 0.20%
Losses train 4.679e+02 val 4.332e+02 test 4.832e+02
```

Figure 26. Performance of configuration V

As it can be seen from Figure 26, the performance of the network increased, the losses during train, validation and testing decrease, meaning that the actual result is close enough to the result predicted by NN.

As a conclusion, the dependency on the settings of the NN are obvious from case to case considered in those chapters. The number of layers, neurons, epochs, batch size as well as type of optimizers are significant while designing the NN. It takes time to find the best combination of parameters in order to train the network properly. At the same time, it is important to avoid overtraining because in this case the NN suffers from the lower ability to predict, meaning that finding the balance is crucial part while designing the NN.

4.2 Double pendulum

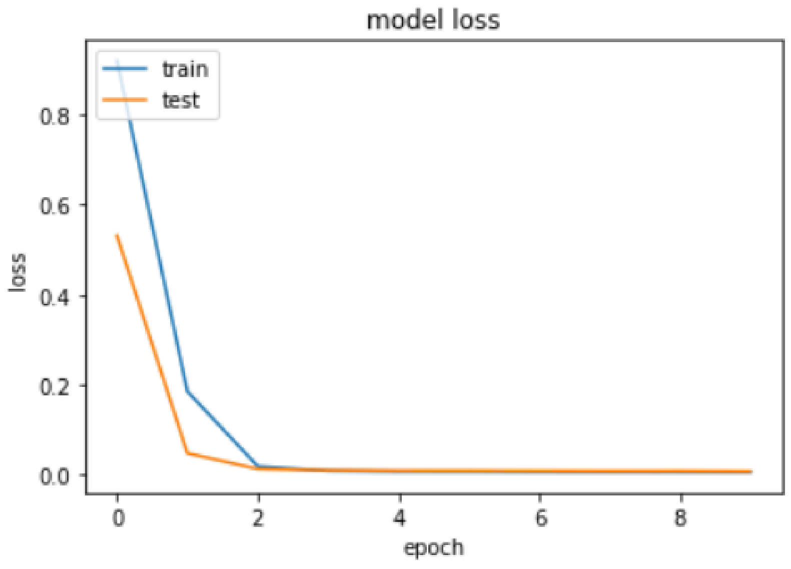
For the double pendulum the NNs were set and tested with the different configurations and settings. This was done to find the optimal configuration of the NN that would give the needed performance.

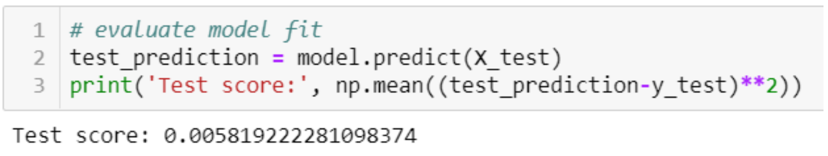
There are three different configurations that were investigated in this part. The results are presented and analyzed below.

4.2.1 Case I

In this configuration the NN was composed of two inner layers with 12 and 4 neurons respectively. The network was trained for 10 epochs with the batch size of 8.

The results are shown in Figures 27. The losses are decreasing. When the results are visualized in order to compare the behavior of real system with the behavior approximated by NN it is obvious that the configuration is not optimal.





```
# evaluate model fit
test_prediction = model.predict(X_test)
print('Test score:', np.mean((test_prediction-y_test)**2))
```

```
Test score: 0.005819222281098374
```

Figure 27. Loss and test score of configuration I

It can be seen from Figure 28 and Figure 29 that in some periods of time the difference between the real system and NN is significant, especially at the areas where the pendulum changes the direction of movement. Time spent to train the NN was equal to 10:42,13

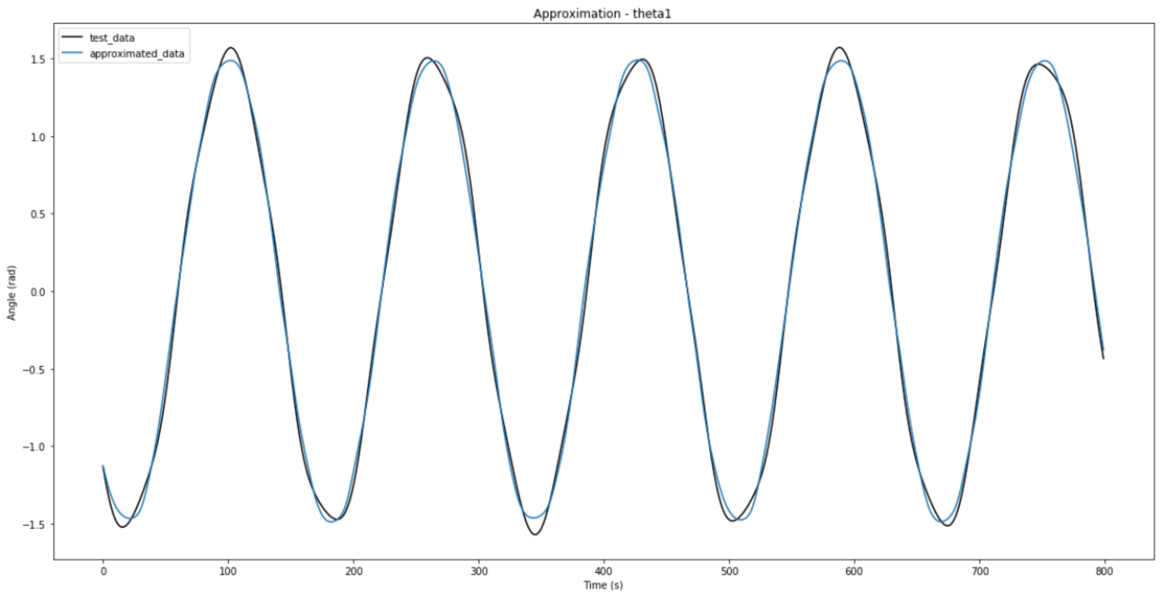


Figure 28. Theta1 approximation vs test data configuration I

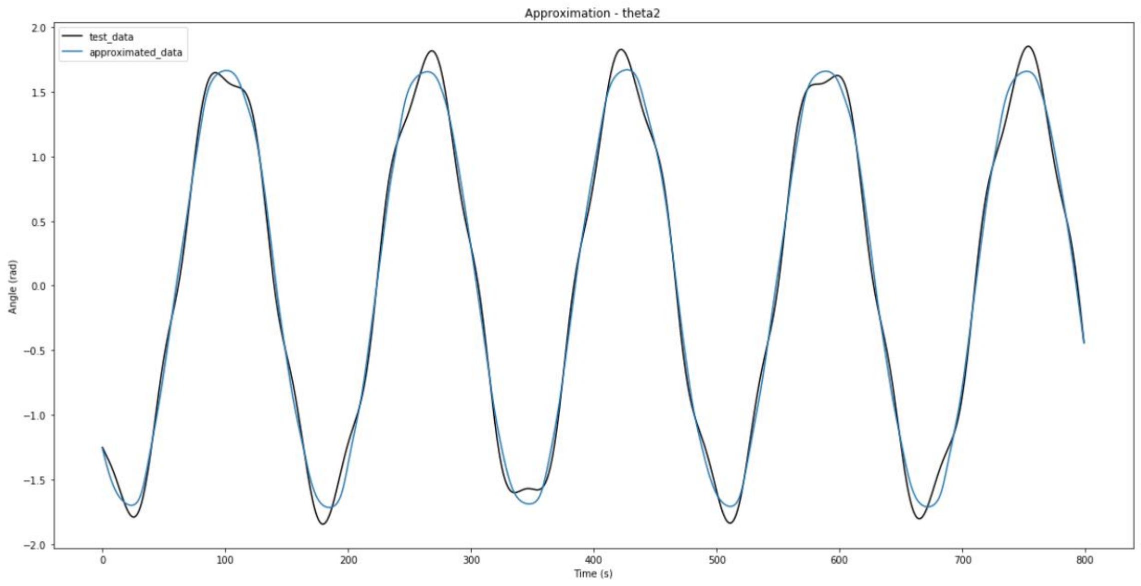
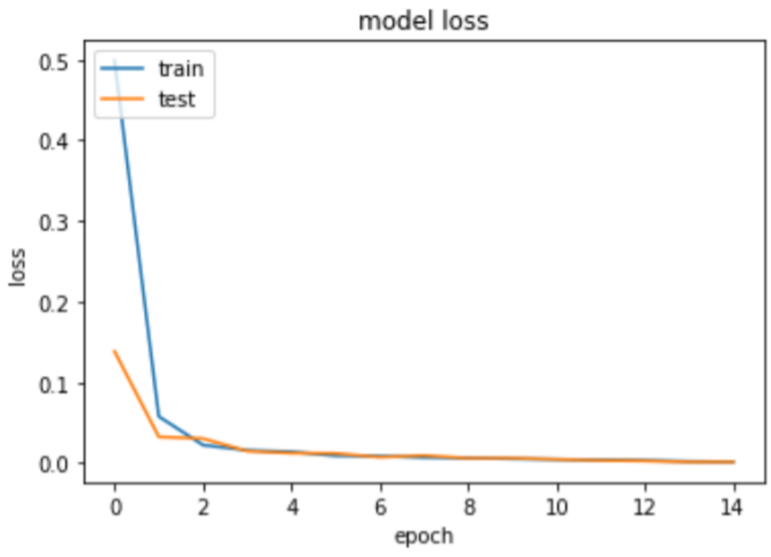


Figure 29. Theta2 approximation vs test data configuration I

4.2.2 Case II

In this case it was decided to increase the number of epochs to 15 and the number of neurons in the first inner layer to 128. The rest parameters stayed the same.

As it can be seen from Figure 30 the overall performance of the system has increased significantly. Test score has improved, and the losses decreased.



```
Test score: 0.001582057303491763
```

Figure 30. Loss and Test Score of configuration II

The behavior of the NN is close to the behavior of the real system. As it is shown in Figure 31 and Figure 32 the graphs are almost identical. There are some insignificant differences at the points where the pendulum is about to change the direction, but the overall performance is better comparing to the first configuration.

Time spent to train the NN was equal to 10:11,79

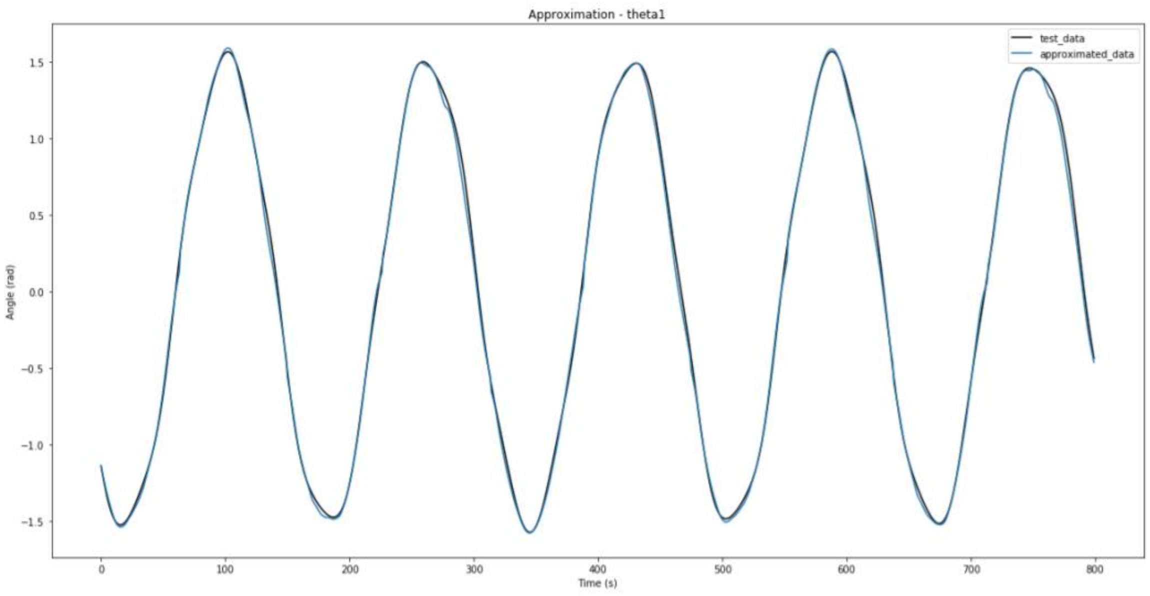


Figure 31. Theta1 approximation vs test data configuration II

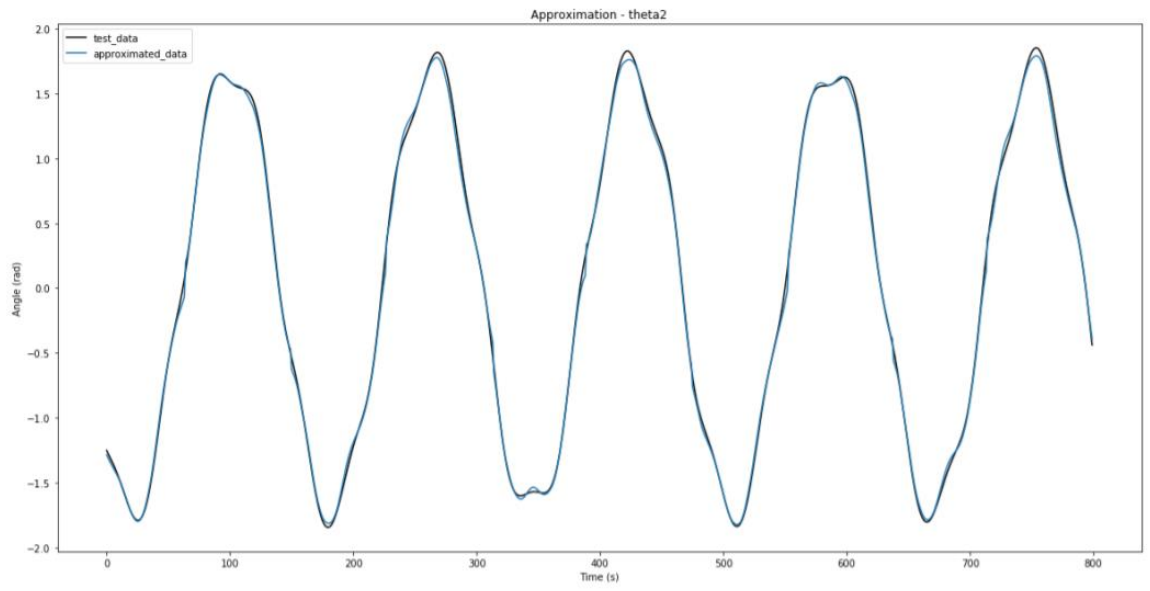
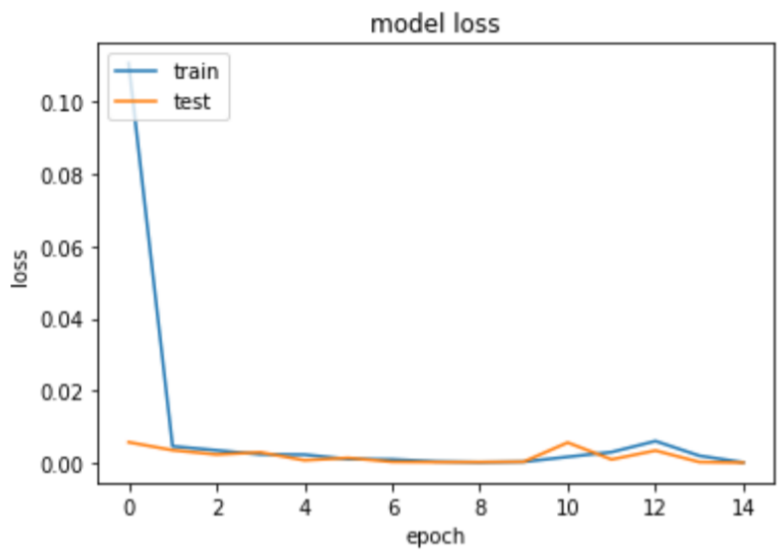


Figure 32. Theta2 approximation vs test data configuration II

4.2.3 Case III

In this case the only change was made inside the first inner layer. The number of neurons was increased up to 256. Figure 33 shows the losses of the configuration and the test score. It is obvious that the configuration gives the better results and test score improves.



```
Test score: 5.783774210943324e-05
```

Figure 33. Loss and Test Score of configuration III

These results are visualized in Figure 34 and Figure 35. It can be seen the movement of the real system is almost completely repeated by the NN. The differences are not significant. The configuration can be positively evaluated. It can be further improved by changing the hyperparameters but in the framework of this project it gives the desired performance.
Time taken to train the NN was equal 10:28,54

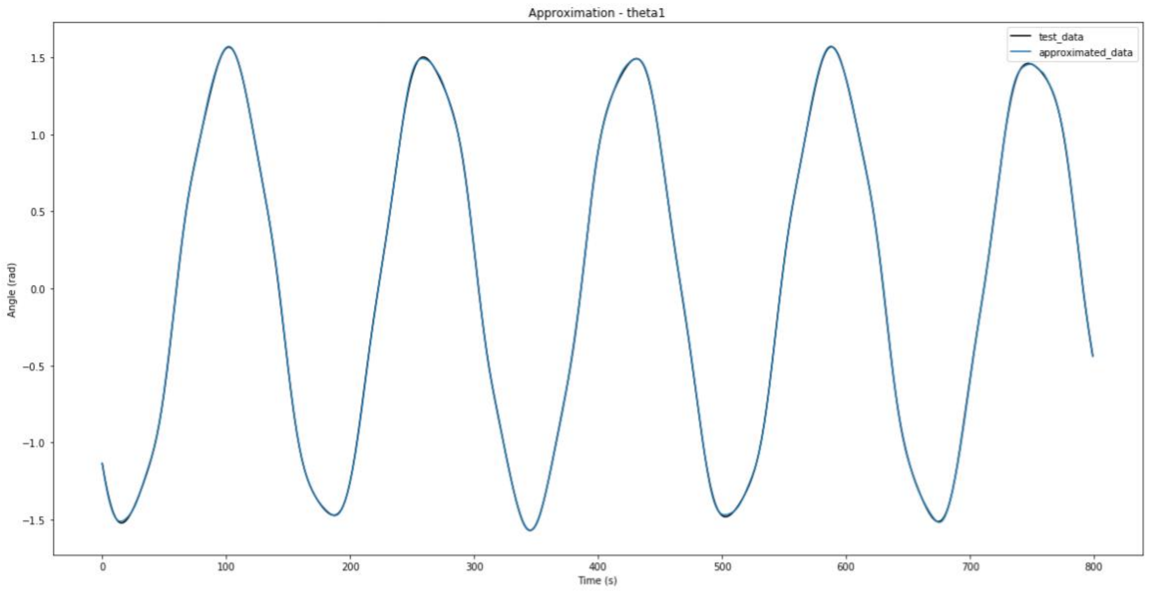


Figure 34. Theta1 approximation vs test data configuration III

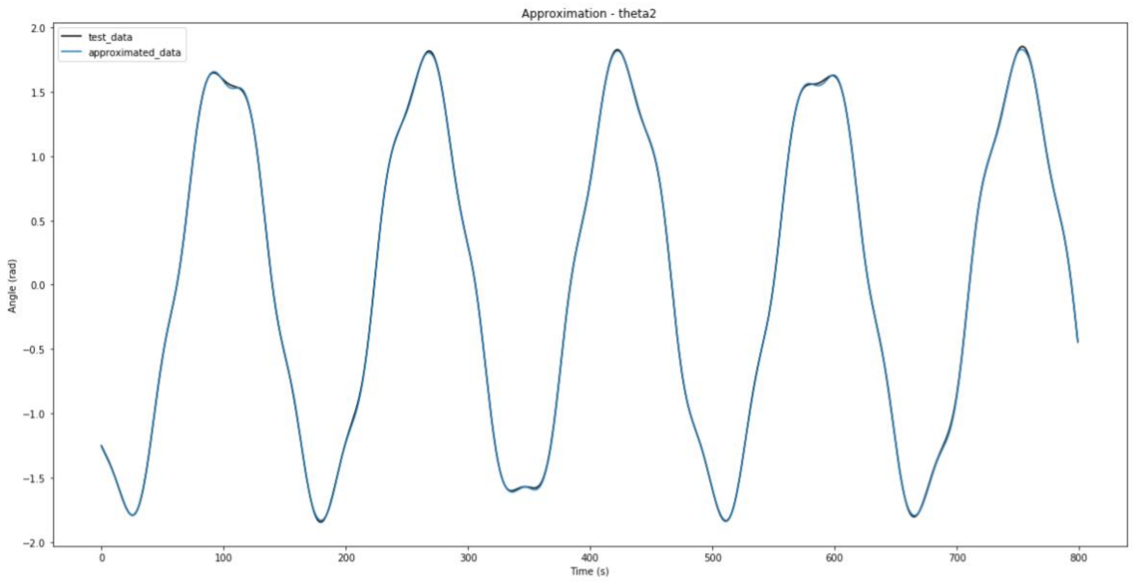


Figure 35. Theta2 approximation vs test data configuration III

5 DISCUSSION

The purpose of discussion is to conduct the analysis obtained in Chapter 4 and do the conclusion about the overall feasibility of Machine Learning in the field of multibody dynamics systems. Firstly, possible application of the technology is described. Secondly, the advantages and disadvantages of the technology are mentioned. In the last part, possible topics and changes of the system, related to the application, are discovered.

5.1 Technology application

The goal of the project is to investigate the feasibility of the application of AI in the field of real-time simulation for heavy machinery. More specifically, it can be used in data-driven design in the framework of multibody system dynamics. By inputting continuously updating data, it is possible to take to the new level of usability of simulation tools.

After training the NN it is possible to get the model which would predict the motion of the object without solving the standard equations. This happens because the network that is trained properly is able to find the patterns in the data it is getting in the real- time. Such technology could be used for predictive design, data analysis and design optimization. In this case it is possible speed up the launching process for a new product, prevent and avoid possible drawbacks in the design in advance. This fact leads to the saving time and money.

5.2 Advantages and drawbacks

The main advantage of two obtained NNs is that the behavior of the models is close enough to the systems which use traditional technology. In the first system, the values of functions are close to the values which were used during the training of network. In the second example the position of double pendulum is close to the values obtained from the equations of motion.

The next advantage is dealing with the fact that the same configuration of the NN can be utilized for the system with different parameters, meaning that the systems is friendly for customization and the only difficulty is to load and define the data. As the result, it is possible to modify the system fast and get the outputs of the system rapidly.

Nevertheless, there is a number of drawbacks which can negatively influence the utilization of the technology. The main problem is that the NN model is still subjected to the imprecisions. It is difficult to catch the edge in order to get the properly working model giving a desired accuracy and to overcome the overfitting, it is important to find the balance. The next problem is dealing with the amount of data. By providing the huge amount of data to process, the computational capabilities can struggle due to the amount of data. Also, the process can slow down by the configuration of NN, for example, by the excessive number of layers or neurons, or by number of epochs used for training the network.

Then, the complexity of the investigated system is able to influence the utilizing prospects of the technology. In the particular research the simplified dynamic systems were studied. This system consists of just several bodies. By complicating the system, the performance of the network can be affected. This should be taken into consideration while evaluating the feasibility of the technology.

The last limitation is due to the fact that the project was limited to the certain types of networks which were used for the systems. For the proper results the other types of NN could be investigated and then evaluated by the performance.

5.3 Suggested improvements and future researches

There is plenty of space for improvement of the current system. This can be dealing with finding the best hyperparameters of the system. To do this the more advanced scripting is needed. By adjusting hyperparameters the more stable and better operating system can be obtained. The next improvement is possible by testing the different types of NNs. Deeper knowledge and analysis are needed to choose the right option for particular case. Moreover, some other NN library can be tested to find the best solution.

Future researches can deal with the improvement of current system and tuning it. Then, it can be adapted for more advanced and industrial applications.

Another research can be aimed at comparative analysis of performances of different types of NN, that later can be used as a reference guide.

6 CONCLUSION

The central objective of the research was to discover the implementation of Artificial Intelligence in the field of multibody dynamics and develop the approach. Then it was needed to evaluate the performances of the networks by comparing them to the real systems and make the conclusions.

Also, the research questions were stated in the introduction. They were:

1. How is possible to utilize AI capabilities instead traditional approaches in multibody dynamics?
2. How to design and build the NN for the systems under consideration?
3. How accurate are the obtained NNs configurations?
4. How long it takes to setup the NNs?
5. What is the computational efficiency of the obtained systems?

All questions were covered in this research. The answer to the main question is going throughout the whole research. The basic theory of NNs was given in the theoretical part, then the practical approach was shown in the research methods where the designing and building processes were shown for two cases: nonlinear system of equations and double pendulum restricted by torsional spring. In these chapters first two questions were covered.

Questions number 3 and 5 were discussed in the chapter dealing with the results. The performances of the Networks are opened in this chapter. During the project the essential task was to check the feasibility of the DL in the field of Multibody Dynamics. The obtained results can be evaluated positively. The obtained NNs are able to replicate the behavior of the systems but much work is needed to investigate the pitfalls influencing the performance.

Question number 4 can be categorized as a subjective question. The time needed to setup the network can vary due to many factors. For example, complexity of the system that is being under consideration. This fact can influence the number of parameters of the system, that causes huge variations in the number of possible inputs and outputs. This difference in the data requires different time for preprocessing the data and finding the best combination of hyperparameters for the system. Moreover, the amount of data that should be fed into the NN has a big influence on the processing, because much data needs more complicated architecture of NN, that leads to longer learning time. Configuration of hardware is also an essential part that is critical for setting time. Powerful hardware is desirable for complex NNs that require tons of computations. Finally, the programming skills of operator are important and significantly influence the time consumption.

Altogether, the overall results of the research can be evaluated as good and positive. The results give a platform for further steps in order to replace the traditional equations-based approaches used in machine dynamics and follow the global trend of the implementation of AI. There is a space to develop existing model and improve by adjusting the hyperparameters that can be also done by developing the programming part of the research. Moreover, it can be used as a reference for more complex systems, consisting of big amount of bodies and components influencing the dynamics of the system.

REFERENCES

Baharudin, M. 2016. Real time simulation of multibody systems with applications for working mobile machines. [web document]. Lappeenranta University of Technology. Available: https://lutpub.lut.fi/bitstream/handle/10024/120728/MEB_A4.pdf?sequence=2&isAllowed=y. Pp. 24-27.

Battini, D. 2018. Adam Optimization algorithms in Deep Learning. [web document] Tech-Quantum Ecosystem. [Referred 17.10.2019]. Available: https://www.tech-quantum.com/adam-optimization-algorithms-in-deep-learning/.

Brownlee, J. 2016. Introduction to Python Deep Learning with Keras. [web document] Machine Learning Mastery. Updated 13.9.2019. [Referred 17.6.2019]. Available: https://machinelearningmastery.com/introduction-python-deep-learning-library-keras/.

Brownlee, J. 2017. What is the Difference Between Test and Validation Datasets?. [web document] Machine Learning Mastery. Updated 27.7.2017. [Referred 26.5.2019]. Available: https://machinelearningmastery.com/difference-test-validation-datasets/.

Chollet, F. 2017. Deep Learning with Python. Manning Publications. Pp. 14, 30-31.

Flores, P. 2015. Concepts and Formulations for Spatial Multibody Dynamics. Berlin, Germany: Springer International Publishing. Pp. 1-2.

Keras. n.d.. Optimizers - Keras Documentation. [web document]. Keras. [Referred 12.10.2019]. Available: https://keras.io/optimizers/.

Michelucci, U. 2018. Applied Deep Learning. Apress. Pp. 31-32, 34, 83, 115.

NVIDIA Developer. 2019. Recurrent Neural Network. [web document]. NVIDIA Corporation. [Referred 12.6.2019]. Available: https://developer.nvidia.com/discover/recurrent-neural-network.

Pattanayak, S. 2018. Pro Deep Learning with TensorFlow. Berkeley, CA: Apress. Pp. 1, 56, 65.

Patterson, J. and Gibson, A. 2017. Deep learning. Sebastopol, California: O'Reilly. Pp. 27-28, 54-55, 65-71, 73-74, 143.

Rosebrock, A. 2017. Deep Learning for Computer Vision with Python. PyImageSearch. p. 127.

Salvaris, M., Dean, D. and Tok, W. 2018. Deep learning with Azure. Berkeley, USA: Apress. Pp. 1,3-4, 9, 15, 25.

Shaikh, F. 2019. Understanding Recurrent Neural Networks (RNNs) from Scratch. [web document]. Analytics Vidhya. [Referred 5.6.2019]. Available: https://www.analyticsvidhya.com/blog/2019/01/ fundamentals-deep-learning-recurrent-neural-networks-scratch-python/ .

Sinha, N. 2018. Understanding LSTM and its Quick Implementation in Keras for Sentiment Analysis. [web document]. Towards Data Science. [Referred 13.6.2019]. Available: https://towardsdatascience.com/understanding-lstm-and-its-quick-implementation-in-keras-for-sentiment-analysis-af410fd85b47.

Trask, A. 2019. Grokking Deep Learning. Manning Publications. Pp. 12-13.

Wei En, J. 2019. How to build your first Neural Network to predict house prices with Keras. [web document]. Free Code Camp. [Referred 4.7.2019]. Available: https://www.freecodecamp.org/news/how-to-build-your-first-neural-network-to-predict-house-prices-with-keras-f8db83049159/.

Yun, Y. 2016. Lagrange Equation by MATLAB with Examples. [web document]. Youngmok Yun: Roboticist in Austin. [Reffered 31.3.2019]. Available: http://youngmok.com/lagrange-equation-by-matlab-with-examples/.

Lagrange equations in Matlab for Double Pendulum. Authors: Youngmok Yun, Denis Bobylev

```
clear;close all;clc;
syms m1 m2;
syms theta1 theta1d theta1dd theta2 theta2d theta2dd;
syms L1 L2 I1 I2;
syms k g
syms y1 y2 y3 y4

x1 = L1/2*sin(theta1);
y1 = -L1/2*cos(theta1);
x2 = L1*sin(theta1)+L2/2*sin(theta2);
y2 = -L1*cos(theta1)-L2/2*cos(theta2);

x1d = L1/2*cos(theta1)*theta1d;
y1d = L1/2*sin(theta1)*theta1d;
x2d = L1*cos(theta1)*theta1d+L2/2*cos(theta2)*(theta2d);
y2d = L1*sin(theta1)*theta1d+L2/2*sin(theta2)*(theta2d);

T = 0.5*m1*( x1d^2 + y1d^2) + 0.5*m2*( x2d^2 +
y2d^2)+1/2*I1*theta1d^2+1/2*I2*theta2d^2;
T=simplify(T);

V = m1*g*y1 + m2*g*y2+k/2*(theta2-theta1)^2;
V=simplify(V);

L = T - V;

pLx1d = diff(L,theta1d);
ddtpLx1d = diff(pLx1d,theta1)*theta1d+ ...
             diff(pLx1d,theta1d)*theta1dd+ ...
             diff(pLx1d,theta2)*theta2d + ...
             diff(pLx1d,theta2d)*theta2dd;
pLx1 = diff(L,theta1);
```

```
pLx2d = diff(L,theta2d);
ddtpLx2d = diff(pLx2d,theta1)*theta1d+ ...
             diff(pLx2d,theta1d)*theta1dd+ ...
             diff(pLx2d,theta2)*theta2d + ...
             diff(pLx2d,theta2d)*theta2dd;
pLx2 = diff(L,theta2);

eqtheta1 = simplify( ddtpLx1d - pLx1);
eqtheta2 = simplify( ddtpLx2d - pLx2);

Sol = solve(eqtheta1,eqtheta2,theta1dd,theta2dd);
Sol.theta1dd = simplify(Sol.theta1dd);
Sol.theta2dd = simplify(Sol.theta2dd);

syms y1 y2 y3 y4
eqM1=Sol.theta1dd
eqM2=Sol.theta2dd
```

Lagrange equations in Matlab for Double Pendulum. Authors: Youngmok Yun, Denis Bobylev

```
function [yprime] = mypend(t, y)

L1=0.4; L2=0.6 ; I1= 0.042018498232; I2=0.01674173303; m1 = 2.6892713258
;
m2=0.5367373941; g=9.80665; k=18/pi;

y_prime=zeros(4,1);



yprime(1) = y(3); %theta1d
yprime(2)=  y(4); %theta2d
yprime(3)= -(2*(8*I2*k*y(1) - 8*I2*k*y(2) + 2*L2^2*k*m2*y(1) -
2*L2^2*k*m2*y(2) + L1*L2^3*m2^2*y(4)^2*sin(y(1) - y(2)) +
L1*L2^2*g*m2^2*sin(y(1)) + L1*L2^2*g*m2^2*sin(y(1) - 2*y(2)) +
4*I2*L1*g*m1*sin(y(1)) + 8*I2*L1*g*m2*sin(y(1)) +
L1^2*L2^2*m2^2*y(3)^2*sin(2*y(1) - 2*y(2)) + 4*L1*L2*k*m2*y(1)*cos(y(1) -
y(2)) - 4*L1*L2*k*m2*y(2)*cos(y(1) - y(2)) + L1*L2^2*g*m1*m2*sin(y(1)) +
4*I2*L1*L2*m2*y(4)^2*sin(y(1) - y(2))))/(16*I1*I2 + 4*L1^2*L2^2*m2^2 +
4*I2*L1^2*m1 + 4*I1*L2^2*m2 + 16*I2*L1^2*m2 + L1^2*L2^2*m1*m2 -
4*L1^2*L2^2*m2^2*cos(y(1) - y(2))^2); %theta1dd
yprime(4)= (2*(8*I1*k*y(1) - 8*I1*k*y(2) + 2*L1^2*k*m1*y(1) -
2*L1^2*k*m1*y(2) + 8*L1^2*k*m2*y(1) - 8*L1^2*k*m2*y(2) +
2*L1^2*L2*g*m2^2*sin(2*y(1) - y(2)) + 4*L1^3*L2*m2^2*y(3)^2*sin(y(1) -
y(2)) - 2*L1^2*L2*g*m2^2*sin(y(2)) - 4*I1*L2*g*m2*sin(y(2)) +
L1^2*L2^2*m2^2*y(4)^2*sin(2*y(1) - 2*y(2)) + L1^2*L2*g*m1*m2*sin(2*y(1) -
y(2)) + L1^3*L2*m1*m2*y(3)^2*sin(y(1) - y(2)) +
4*L1*L2*k*m2*y(1)*cos(y(1) - y(2)) - 4*L1*L2*k*m2*y(2)*cos(y(1) - y(2)) +
4*I1*L1*L2*m2*y(3)^2*sin(y(1) - y(2))))/(16*I1*I2 + 2*L1^2*L2^2*m2^2 +
4*I2*L1^2*m1 + 4*I1*L2^2*m2 + 16*I2*L1^2*m2 + L1^2*L2^2*m1*m2 -
2*L1^2*L2^2*m2^2*cos(2*y(1) - 2*y(2))); %theta2dd

yprime=yprime';

end
```

Lagrange equations in Matlab for Double Pendulum. Authors: Youngmok Yun, Denis Bobylev

```
%-----------------------------------------------------------------------
--
%-------------- Double Pendulum-----------------------------------------
--
%-----------------------------------------------------------------------
--

clc
close all
clear all;

%---------Parameters----------------------------------------------------
-

L1=0.4; L2=0.6 ; I1= 0.042018498232; I2=0.01674173303; m1 = 2.6892713258
;
m2=0.5367373941; g=9.80665; k=18/pi;

%---------initial condition---------------------------------------------
-


init = [1.57079632679; 1.57079632679; 0; 0];

[t,y]=ode45(@SymbolicSolution1, [0:0.01:10],init);


%---position of mass 1 and mass 2----------------------------------------

x1=L1/2*sin(y(:,1));
y1=-L1/2*cos(y(:,1));
x2=L1*sin(y(:,1))+L2/2*sin(y(:,2));
y2=-L1*cos(y(:,1))-L2/2*cos(y(:,2));

% ------visualizing the result------------------------------------------
--

%plot(t, x1, t, adamspendulumresults.VarName2)


   figure(1)
   plot(t,x1,t,x2)
   hold on; grid on
   title('Double pendulum')
   xlabel('Time [s]')
   ylabel('Position')
   legend('X1')
   %legend('Theta1 [rad]', 'Theta2 [rad]', 'Theta1d [rad/s]', 'Theta2d
[rad/s]')
```