Joint Institute for Nuclear Research

Laboratory of Informational Techonologies

Final report of the summer student programm

Research of the possibilities of using artificial neural networks for calculating n-dimensional integrals

Student:	**Melnitskii Dmitrii**
Supervisor:	**Hovik Grigorian**
Period:	**30 Jule - 6 August**

Dubna, Russian Federation
2017

Contents

1. Abstract

Nowadays we spend a lot of time for calculating n-dimensional integrals, using different methods, like Monte-Carlo and Gauss methods. Computations by these methods expends a lot of computer resources, which are leading to the problem of golden mean between computing speed and resource employment. I and my supervisor developed and researched a method of calculation which is based on artificial neural networks.

2. Test

In our research work we studied a new method of calculating n-dimensional integrals, which is based on artificial neural networks(ANN).

Motivation of this research was that nowadays neural networks are very usable and there are no so many methods of calculating n-dimensional integrals.

In our method, as neural network, we use preceptron, because of his simplicity.

The biggest problem, when you are dealing with ANN, is the learning of this network. At first, we had used the backpropagation method for training, but, as usuall it happens with simple methods, we realized that this method is too slow for our goals that's why we started using Levenberg–Marquardt algorithm, which is much more faster, but it requires a lot of memory. Of course, there are some methods, which are slower then Levenberg–Marquardt algorithm, faster then backpropagation method and don't requires such a big quantity of memory.

The result of our work is that our ANN can calculate 1-dimensional, 2-dimensional, 3-dimensional, 4-dimensional and 5-dimensional integrals of any continuous function on a tilted interval. At this moment, we are trying to modificate our programme for calculating any 5-dimensional integrals of continuous function on a tilted interval with a good accuracy.

Also, i want to note that our shceme allows to calculate any n-dimensional integral.

2.1. Artificial Neural Networks

Artificial neural networks are computing systems inspired by the biological neural networks that constitute animal brains. ANN consists of 3 parts:

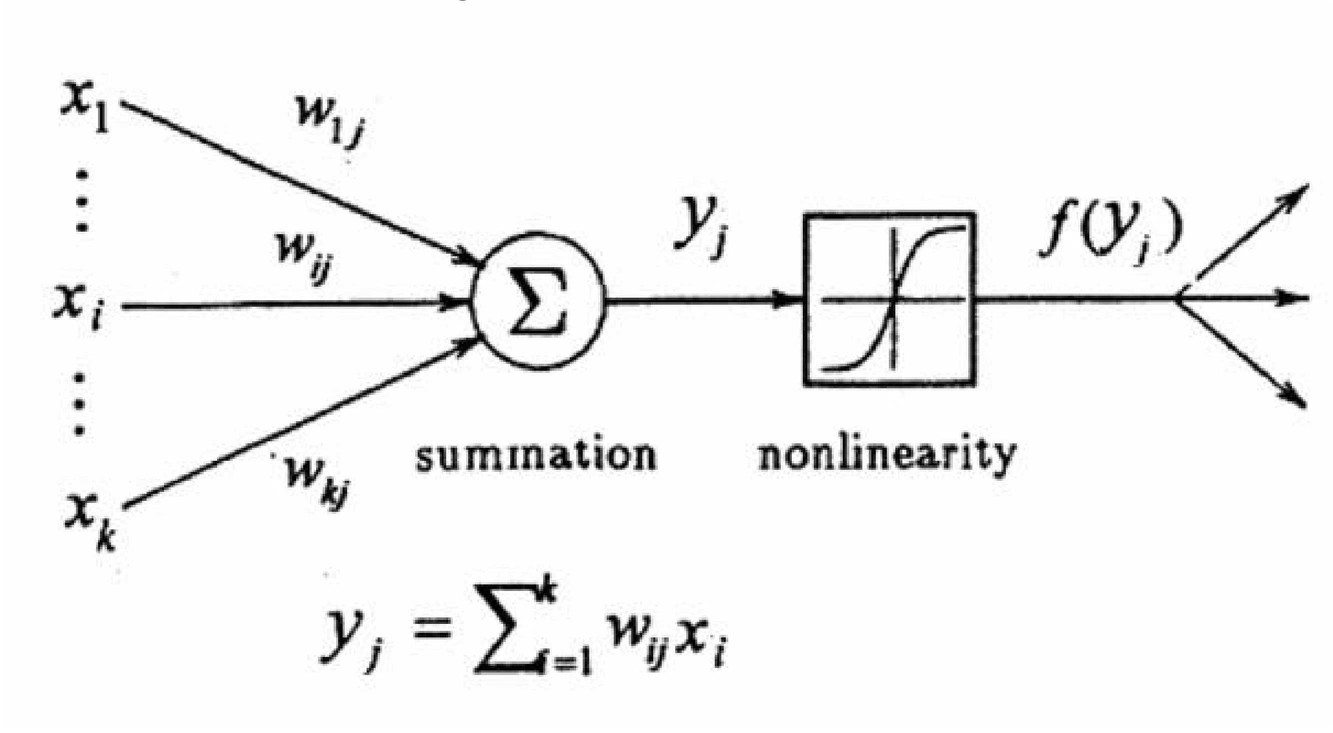
- Neurons
- Linkages between neurons
- Activation functions
- Weights and biases

We are going to have a small excurus into this parts of ANN.

2.1.1. Neurons

Neuron in ANN - mathematical model of biological neuron. We have

Figure 1. Scheme of neuron



a lot of input connections of our neuron. Each of them gives a signal to neuron, which is miltiplied by weight, different for each neuron. After that, all signals are summed up.

$$Input = \sum_{i=1} w_i * signal_i \tag{1}$$

Afterwards, that signal arrives at activation function, which outcome is Output of our neuron, which is going to other neurons in the following connections.

$$Output = f(Input + bias) \tag{2}$$

2.1.2. Linkages

There are a few types of linkages between neurons. Most important and famous of them are multylayer perceptrons and Hopfield networks. In perceptrons there are, at least, 2 layers. Signal goes from input layer

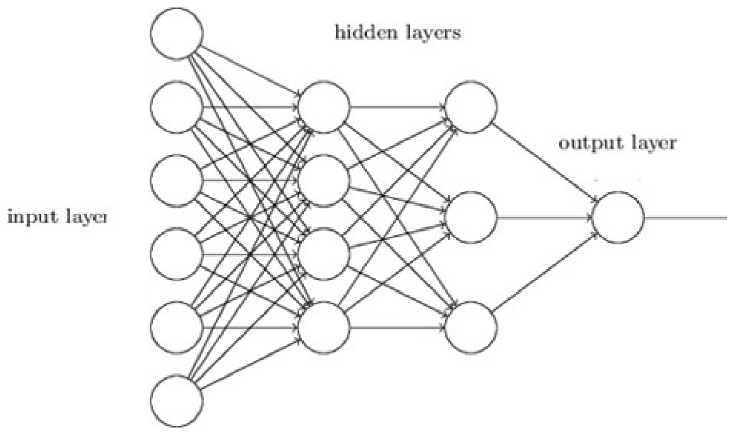


Figure 2. Perceptron

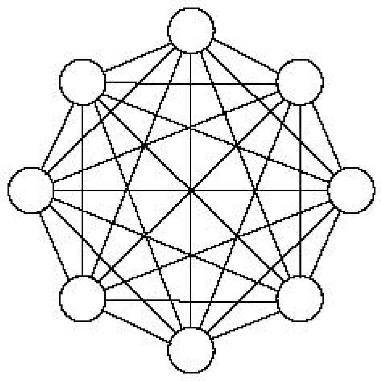


Figure 3. Hopfield network

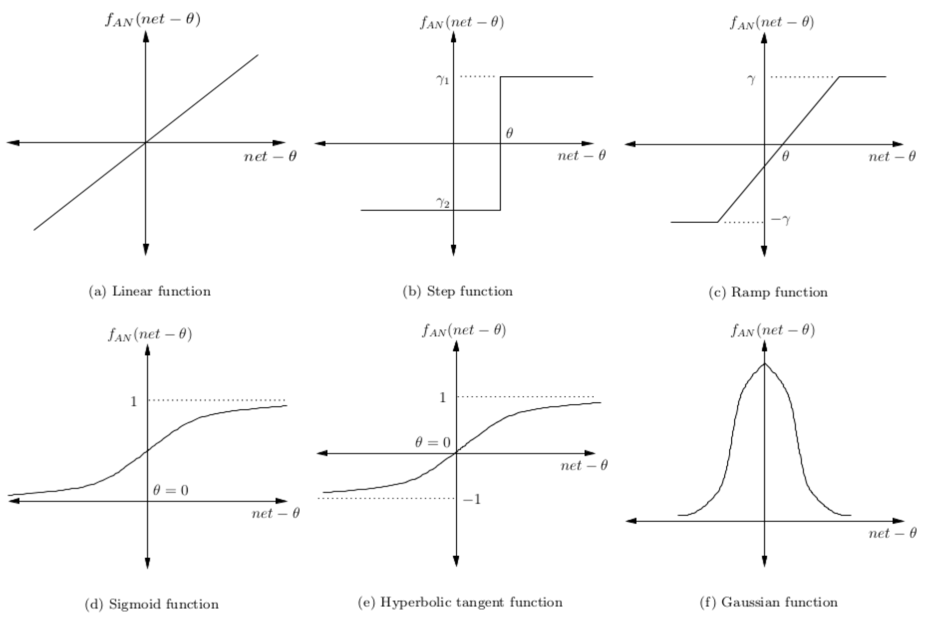
to output layer through hidden layers. The critical moment is that signal can not goes back and can't jumped through one layer.

In Hopfield networks each neuron connected to all others, that's why it is fully another case.

2.1.3. Activation function

We can say that the main target of neuron is to transform the input signal in a special way. It is understood that there are different types of activation functions. Every type has it's own props and cons. For ex-

Figure 4. Different types of activation functions



ample, the most frequently used function is sigmoid, because it increases a weak signal and weakens strong. Gaussian function usually uses in Radial basis function network, which is a separate interesting theme.

In our work we used 2 types of activation functions: sigmoid and linear functions.

2.2. Learning of ANN

Learning of ANN is a learning with a teacher. It means, that we have a table of figures, which consists of 2 parts: inputs and outputs.

At first, we generate random weights and biases in our ANN, after that we simply loop through all of our table input figures and look at the result. This result we compare with table output and make error function.

$$H = \frac{1}{2n} \sum_{alloutputs} (Result - OutputTable)^2 \tag{3}$$

Our goal is to change our weights and biases in the way, that will reduce our Error function, so we have a problem of finding minimum of function. There are some methods, which are most common for ANN:

- Backpropagation
- Levenberg-Marquardt algorithm
- Bayesian regularisation
- Scaled Conjugate Gradient

Some explanations about this methods you can find below in section "Methods".

2.3. Approximation of continuous functions

2.3.1. Rationale

We can legitimately say that any continuous function could be approximate with any precision by a perceptron, because of the theorem, which is presented below.

Theorem. *An arbitrary continuous function, defined on* $[0,1]$ *can be arbitrary well uniformly approximated by a multilayer feed-forward neural network with one hidden layer (that contains only finite number of neurons) using neurons with arbitrary activation functions in the hidden layer and a linear neuron in the output layer.*[1]

$$\sqsupset \varphi(.) - arbitrary\ activation\ function.\ Then\ \forall f \in C\left([0,1]\right) \tag{4}$$

$$\forall \varepsilon > 0\ \exists n_0 \in N,\ \exists w_{1j},\ \exists w_i : \sup_{x\in[0,1]} \left| f(x) - \sum_{i=0}^{n} w_i \cdot \varphi(w_{1i} + v_i) \right| < \varepsilon$$

This theorem says, that we can find such values, as n_0, w_{ij}, w_i, where n_0 - number of neurons in the hidden layer of perceptron, w_i and w_{ij} - weigths of the neural network, and approximation error won't be higer then a given ε.

So, we can conclude, that any continuous function can be approximated by a perceptron with 1 hidden layer with any given precision, and this is another reason why we chose this type of neural network.

2.3.2. Examples

Here there are some examples of approximation of continuous function of 1 and 2 variables. I would like to note that the time of training ANN in this examples stands at 1 second. Stopping conditions were: 1000 iterations or gradient of learning smaller then 10^{-7}. Of course, we can change our stopping conditions in any way that we want. More examples are in the point "Figures".

[1]"Approximation with Artificial Neural Networks","MSc thesis", Balázs Csanád Csáji

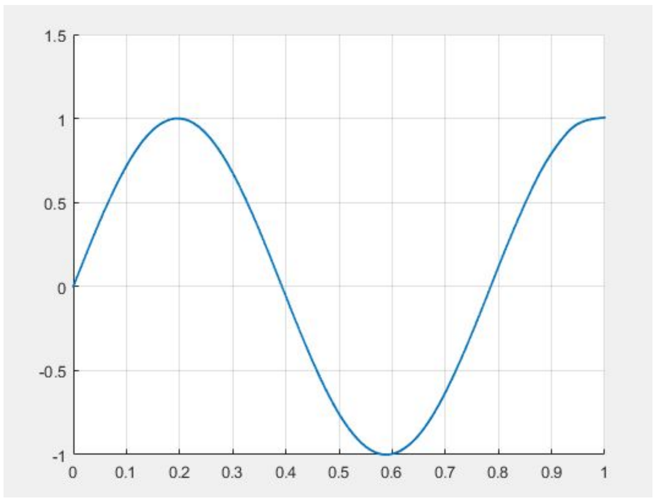


Figure 5. Plot of $sin(8x)$

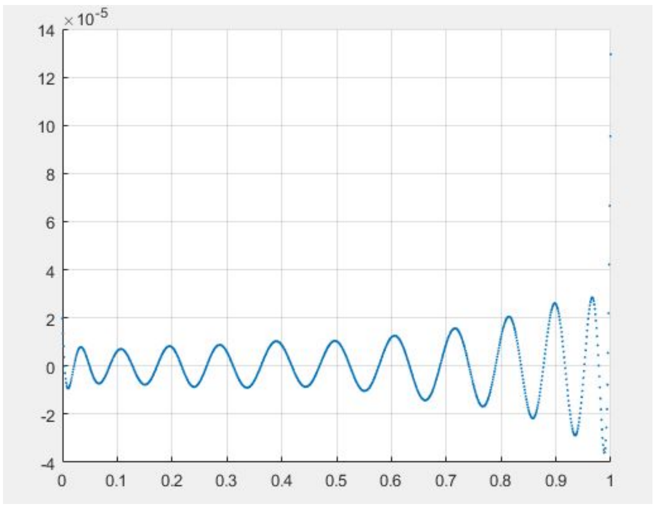


Figure 6. Error of approximation

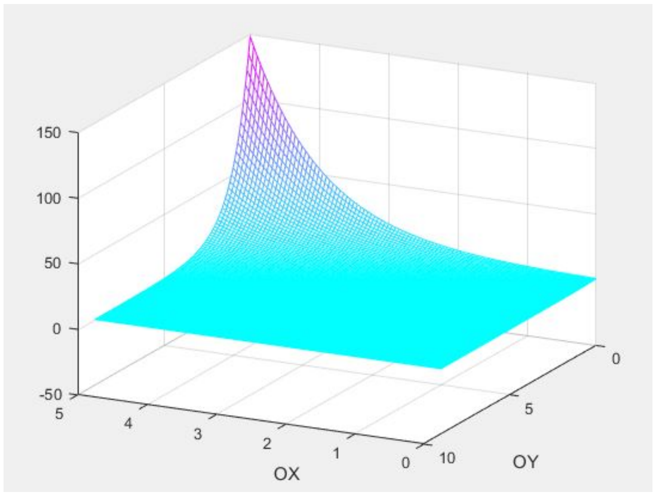


Figure 7. Plot of $exp(x - y)$

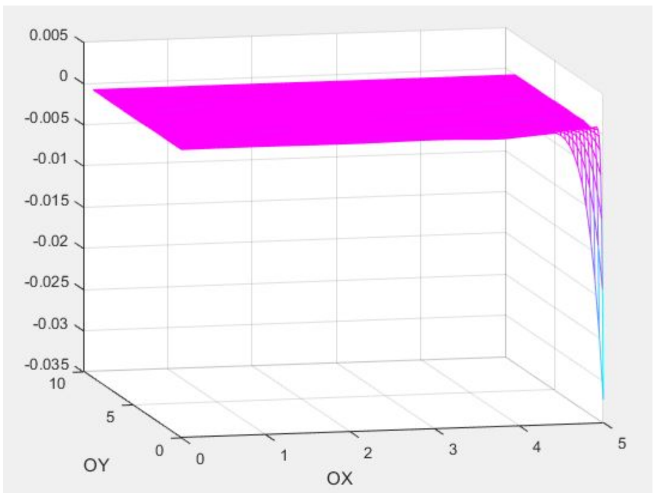
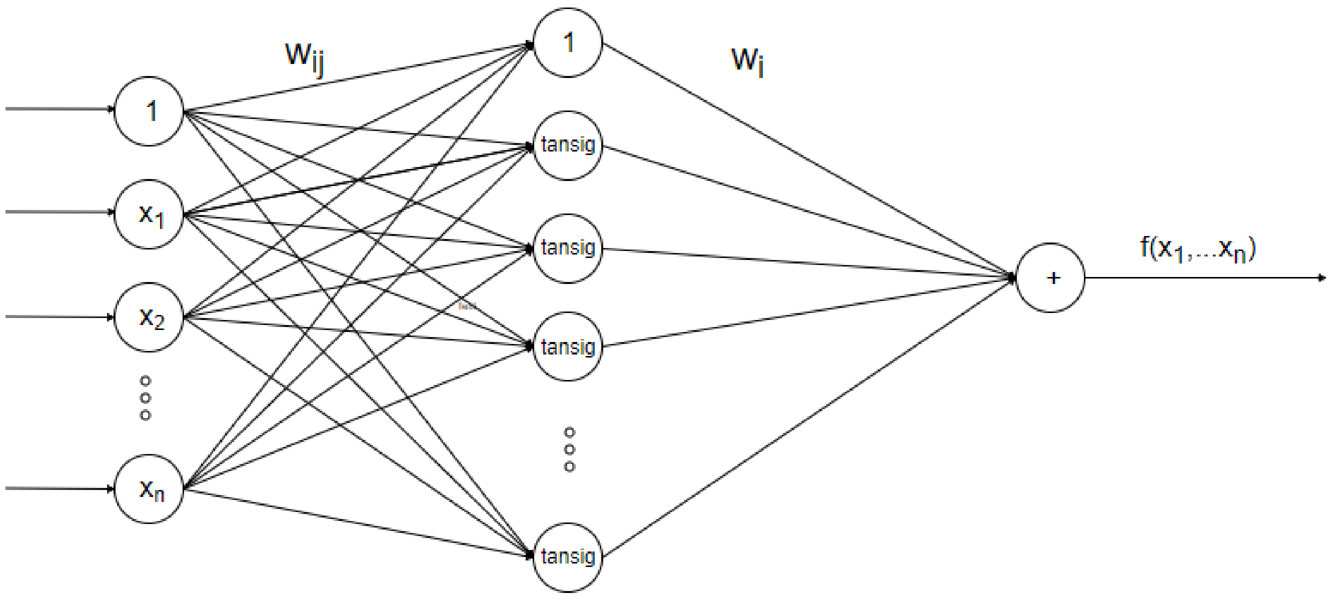


Figure 8. Error of approximation

2.4. Scheme of our neural network

Figure 9. Sheme of neural network



On the Figure 9 you can see the scheme of our neural network, which is used for calculating n-dimensional integrals. It's a perceptron with one hidden layer. As activation functions in the hidden layer we chose Hyperbolic tangent sigmoid trasnfer function, or:

$$\varphi(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

Benefits of this activation function are that this function is infinitely differentiable and learning process occurs fast. On the output layer, as activation function, we took linear function. Reason for our choice will be shown later.

2.5. Scheme of calculating n-dimensional integrals

As it was said earlier, we chose a perceptron with 1 hidden layer, for reasons that we mentioned in the paragraph "Approximation of continuous functions". Let's look at the procedure, which is used in our programme. First of all, we have to approximate our function with the help of neural network. Let's assume that approximation procedure was done. Then we can write:

$$f(x_1, x_2...x_n) \approx Out = \left(\sum_{i=1} \omega_i \cdot \varphi \left(\sum_{j=1} (w_{j,i} \cdot x_j + v_i) \right) + v_{out} \right) \quad (6)$$

Our main target is to integrate this function, and because of 6, we can write:

$$\int f(x_1, x_2, ...x_n) dx_1 \simeq \int \left(\sum_{j=1}^{N} w_j \cdot \varphi \left(\sum_{i=1}^{n} (w_{ij} \cdot x_i + v_i) \right) + v_{out} \right) dx_1 \quad (7)$$

$$\int f(x_1, x_2, ...x_n) dx_1 \simeq \sum_{j=1}^{N} \frac{w_j}{w_{j1}} \cdot \Phi \left(\sum_{i=1}^{n} (w_{ij} \cdot x_i + v_i) \right) + v_{out} \cdot x_1 = \Psi(x_1, x_2, ...$$

$$\Phi(x) = ln(cosh(x))$$

$$\sqsupset \sum_{j=1}^{N} \frac{w_j}{w_{j1}} \cdot \Phi \left(\sum_{i=1}^{n} (w_{ij} \cdot x_i + v_i) \right) + v_{out} \cdot x_1 = \Psi(x_1, x_2, ...x_n) \quad (8)$$

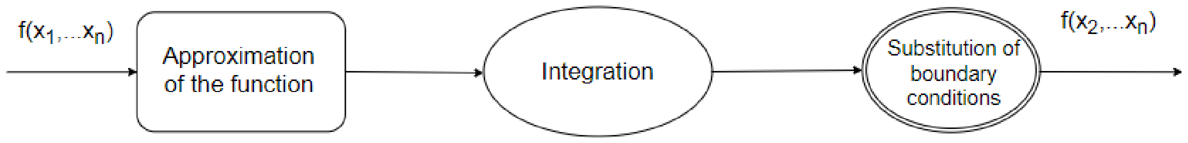
After substitution of boundary conditions we would have next expression:

$$\int_{x_1^{down}(x_2,...x_n)}^{x_1^{up}(x_2,...x_n)} f(x_1, x_2, ...x_n) dx_1 \simeq \Psi(x_1^{up}(x_2, ...x_n)) - \Psi(x_1^{down}(x_2, ...x_n)) \quad (9)$$

As you can see, in the expression 9 we don't have dependence on variable x_1, that's mean that from function of n variables we got to the function of $(n-1)$ variables after the process of approximation, integration and substitution of boundary conditions.

Figure 10 shows the process of getting from a function of n variables to the function of $(n-1)$ variables.

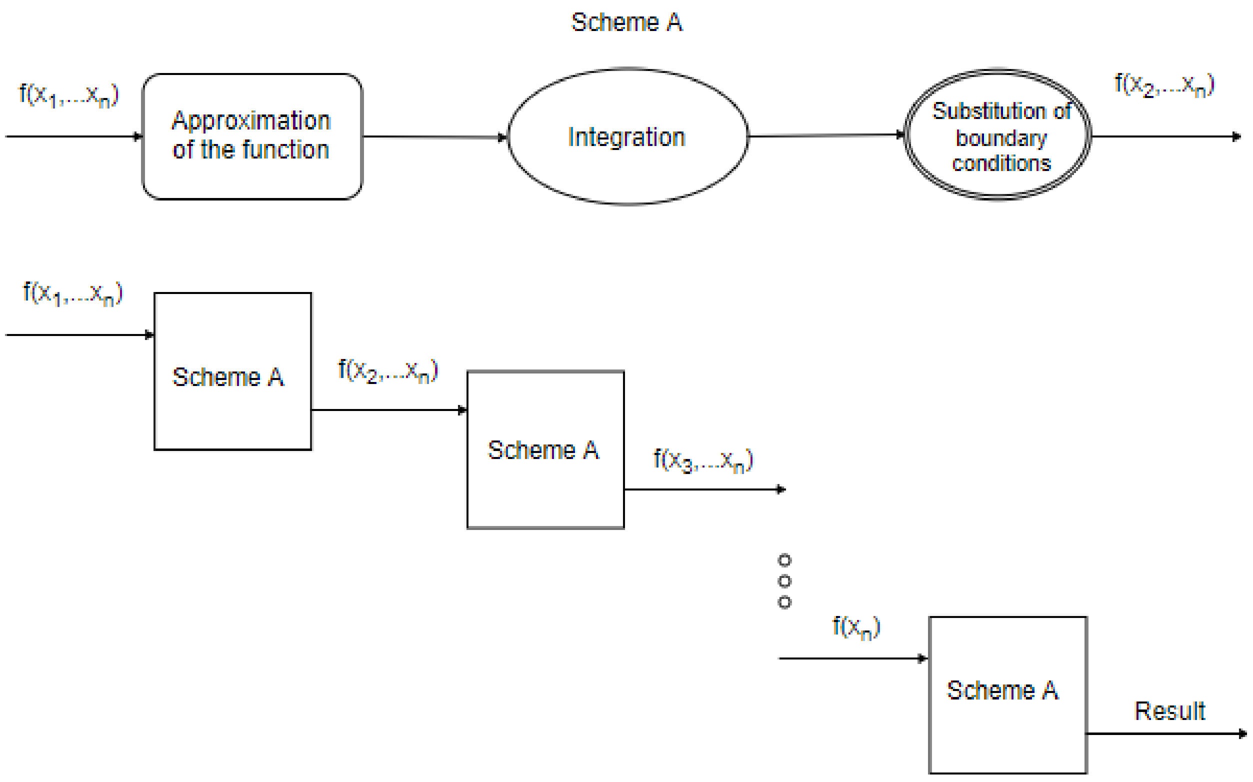
Figure 10. Sheme of one iteration



$f(x_1,...x_n)$
Approximation of the function
Integration
Substitution of boundary conditions
$f(x_2,...x_n)$

The procedure is that we are repeating scheme, that's illustrated on the Figure 10, n times. Schematically, we can represent this procedure in the next way:

Figure 11. Sheme of our method's procedure



Main idea of our procedure, is that we get each integral on each iteration, which is leading us to the result of the problem.

2.6. Examples

2.6.1. 1-dimensional integrals

Let's look at few examples of calculating usual integrals of function of 1 variable.

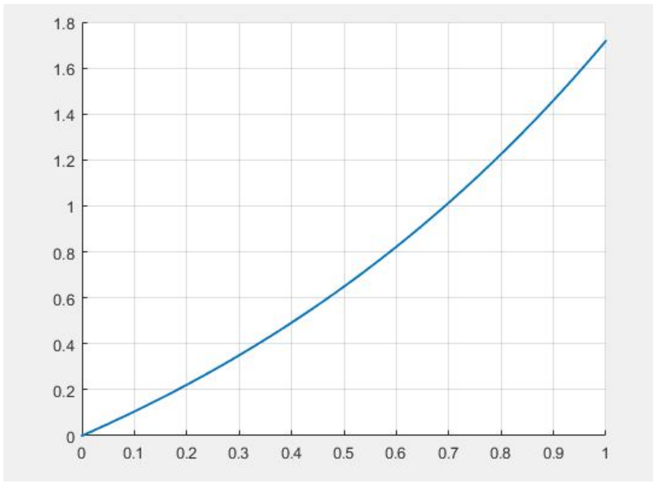


Figure 12. $\int_0^x e^x dx, x' \in [0,1]$

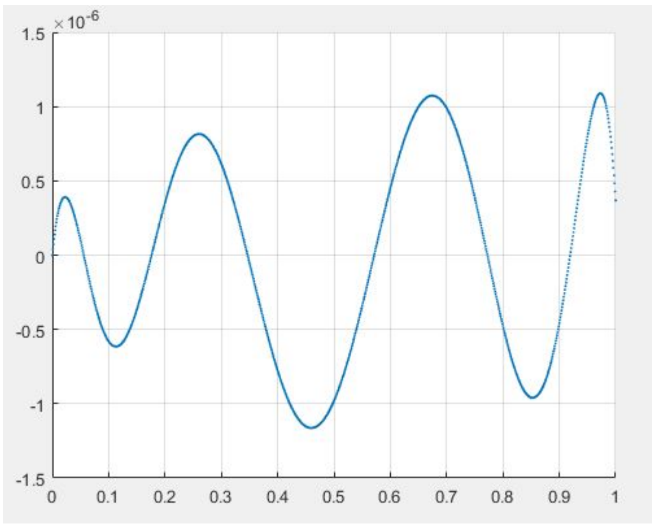


Figure 13. Error of integration

In all examples, learning time is less then 10 seconds.

Also, I'd like to note that in fact there is dependence of quality of learning on number of neurons in our neural network, but we won't face with this problem, because it's very difficult for us at this moment.

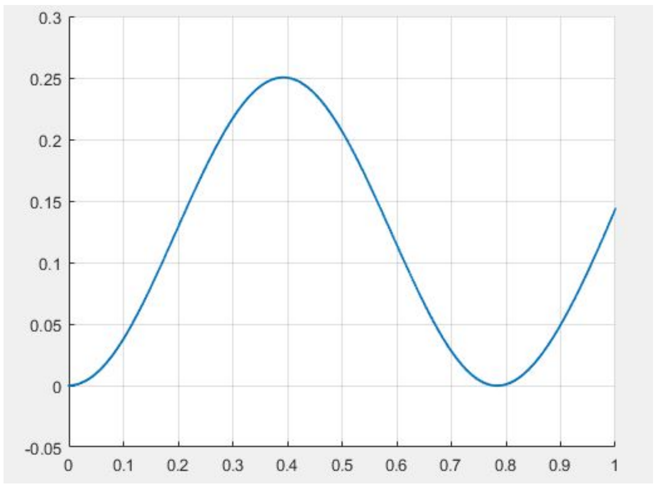


Figure 14. $\int_0^{x'} sin(8x)dx, x' \in [0,1]$

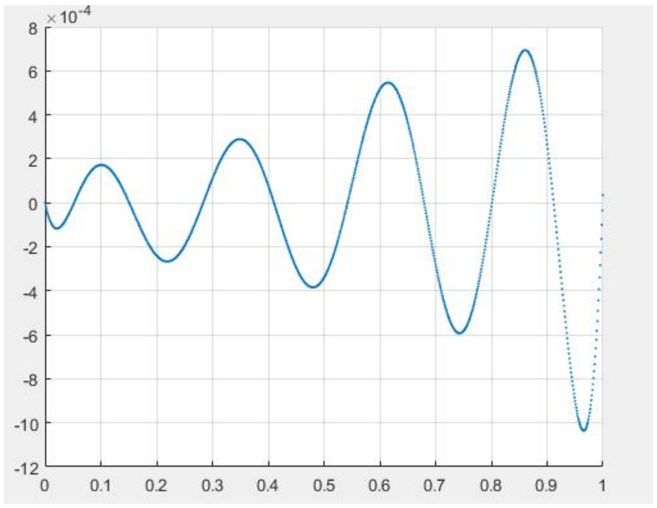


Figure 15. Error of integration

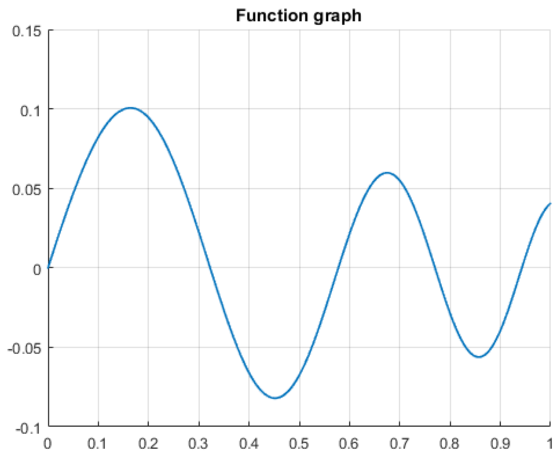


Figure 16. $\int_0^{x'} sin(8exp(x))dx, x' \in [0,1]$

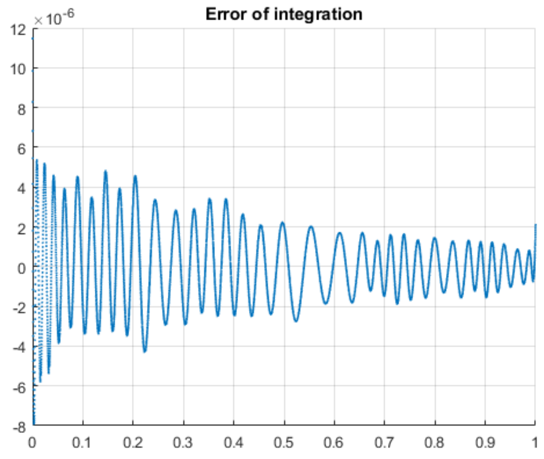


Figure 17. Error of integration

2.6.2. Calculation of two-dimensional integrals

Lets consider next integral:

$$\iint\limits_D f\,dS \tag{10}$$

Let's reckon that $f(x)$ is continuous and D is a limited space area. By using Fubini theorem we can write next:

$$\iint\limits_D f\,dS = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} f(x,y)dy \tag{11}$$

After training of our neural network to the function, $f(x,y)$ we would have expression 6, where we have 2 variables. After that, we can get antiderivative of this function on the variable y:

$$\int_{y_1(x)}^{y_2(x)} f(x,y)dy = F(x) \tag{12}$$

Subsequently, we can create a new neural network, which would be approximate for a $F(x)$.

$$F(x) = \sum_{i=1} \omega_i \cdot \varphi\left(w_i \cdot In + v_i\right) + v_{out} \tag{13}$$

And we can do the same conversion as in the previous paragraph.

$$\hat{F}(x) = \int_a^x F(x')dx' = \int_a^x \left(\sum_{i=1} \omega_i \cdot \varphi\left(w_i \cdot In + v_i\right) + v_{out}\right) dx' \tag{14}$$

And our answer will look like this:

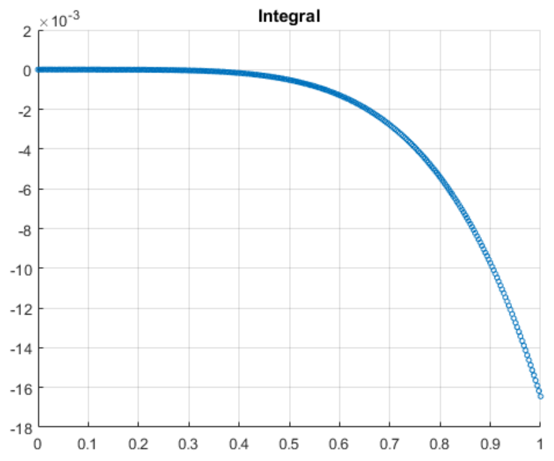
$$\iint\limits_D f\,dS = \int_{x_1}^{x_2} dx \int_{y_1(x)}^{y_2(x)} f(x,y)dy = \hat{F}(x_2) - \hat{F}(x_1) \tag{15}$$

Let's consider next integral.

$$\int_0^y dy' \int_{\frac{y'}{2}}^{y'} f(x,y)dx$$

$$f(x,y) = xy \cdot sin(x-y)$$

$$x, y \in [0,1]$$



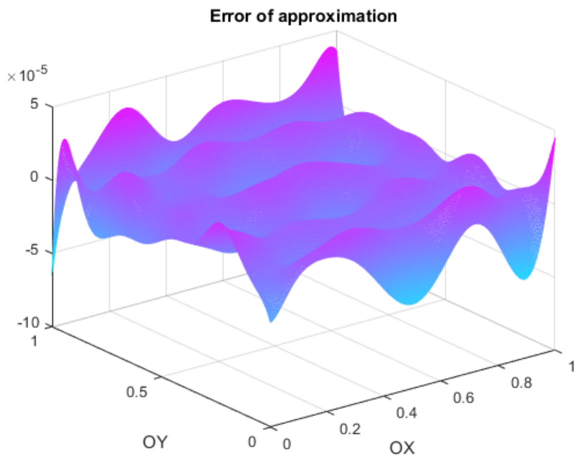


Figure 18. Primitive

Figure 19. Error of integration

More examples of calculations are in the paragraph "Figures". **It's very important to say, that in our examples(Paragraphs 2.6.1,2.6.2) error of approximation coinside with the error of integration, because of the reasons, which are indicated in the paragraph "Methods"**

2.6.3. Calculation of five-dimensional integral

Let's consider next integral.

$$\int_0^3 dx_5 \int_{\frac{x_5}{2}}^{x_5} dx_4 \int_{\frac{x_4}{2}}^{x_4} dx_3 \int_{\frac{x_3}{2}}^{x_3} dx_2 \int_{\frac{x_2}{2}}^{x_2} dx_1 \cdot f(x_1)$$

$$f(x, y) = x_1$$

$$x_1, x_2, x_3, x_4, x_5 \in [0, 3]$$

This integral can be calculated with the help of our method.

Value, calculated by our system: 0.6012
Desired value: $\simeq 0.6034$

As we can see, in this simple example we got comparably good result. Unfortunately, at this moment, we have problem with a memory that's leading to a bad approximation of given function. We're going to test our method on a heterogeneous cluster.

3. Methods

3.1. Learning methods for ANN

The problem of learning ANN is to find minimum of error function(Paragraph 2.2). Most common methods for perceptrons are gradien methods, like backpropagation.

3.1.1. Backpropagation method

Backpropagation method consists of 4 parts:

- Propagation of the signal through the neural network
- Calculation of the error function
- Propagation of the output activations back through the network using the training pattern target in order to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons
- Weight's update, which is depended on the gradient of the error function

Here are the formulas, which are describing the process of weight's updating.

$$w_{ij} = w_{ij} + \triangle w_{ij} \tag{16}$$

$$\triangle w_{ij} = -\eta \frac{\partial H}{\partial w_{ij}} = -\eta o_i \delta_j$$

where η - coefficient of learning, o_i - signal on i neuron, δ_j - error of j neuron

More about this method you can find in the work of R.Rojas "Neural Networks".

3.2. Features of our method

3.2.1. Accuracy on the cube [0,1]

If we have the accuracy of approximation(mean squared error) less then ε, then accuracy of integration won't grow up.

$$\int_a^b f(x)dx = \int_a^b \Psi(x)dx + \int_a^b Error(x)dx$$

$$\int_a^b f(x)dx - \int_a^b \Psi(x)dx = \int_a^b Error(x)dx \leqq \varepsilon \cdot (b-a) = \varepsilon$$

That's the reason, why we are considering functions on the cube $[0,1]$. It's very easy for us to say, what accuracy we would get after our procudere. At this moment, nobody did the full theory of neural network approximation.

3.2.2. Full primitive on the last integral

In the paragraph 2.5. we devised a formula of integral expressed through the neural network:

$$\int_{x_1^{down}(x_2,...x_n)}^{x_1^{up}(x_2,...x_n)} f(x_1,x_2,...x_n)dx_1 \simeq \Psi(x_1^{up}(x_2,...x_n)) - \Psi(x_1^{down}(x_2,...x_n))$$

Let's say, that we are on the last step of our procedure.

$$\int_a^b f(x_n)dx_n \simeq \Psi(b) - \Psi(a)$$

But parameter b can be different, as approximation was done on a full interval $[a,b]$.

$$\int_a^{x'} f(x_n)dx_n \simeq \Psi(x') - \Psi(a) \tag{17}$$

And situation becomes very good, because we know everything about function Ψ. In our opinion, this feature is the most important in this method.

4. Figures

4.1. Approximation of continuous functions

In this section we will see some examples of approximation with the help of neural networks.

Let's consider next function.

$$f(x) = e^{sin(10x)} \cdot x$$

$$x \in [0, 1]$$

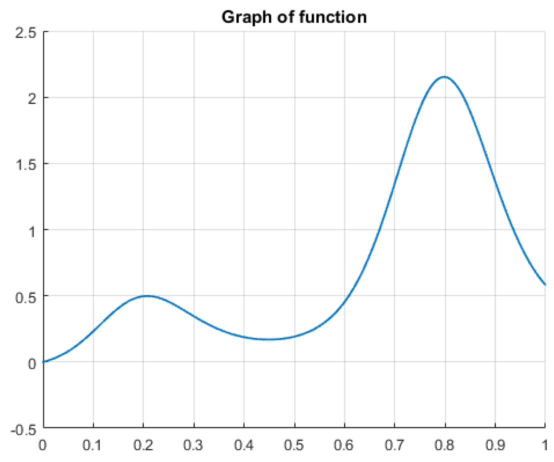


Figure 20. Graph of function

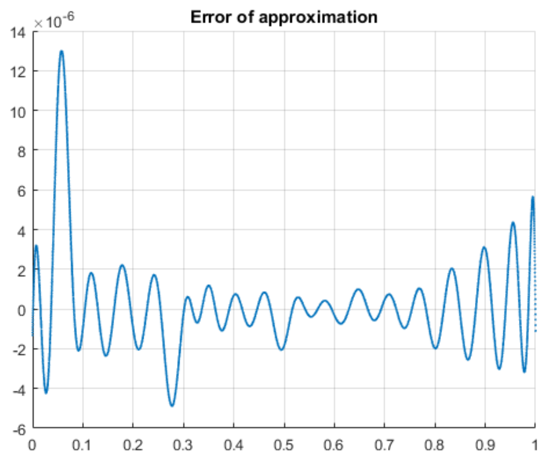


Figure 21. Error of approximation

- Number of neurons: 15
- Elapsed time: $12s$
- Number of approximation points: 100

Let's consider same function but with different number of approximation points.

$$f(x) = e^{sin(10x)} \cdot x$$

$$x \in [0, 1]$$

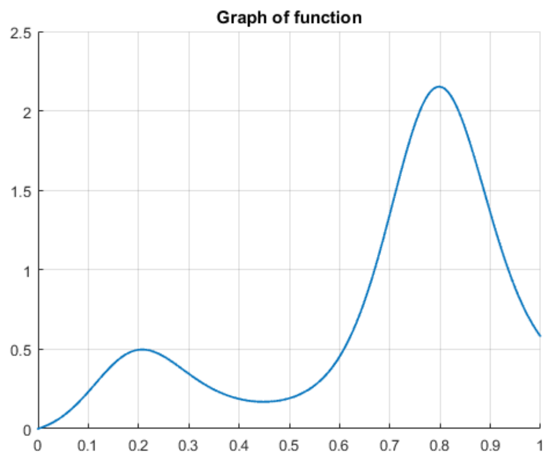


Figure 22. Graph of function

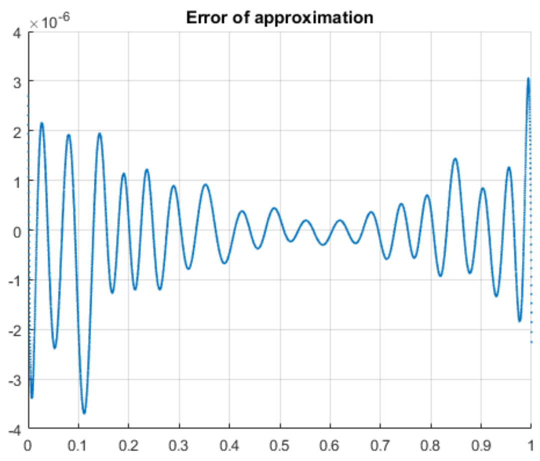


Figure 23. Error of approximation

- Number of neurons: 15
- Elapsed time: $12s$
- Number of approximation points: 300

Difference between these two example is that oscillations of error are lower on the second example, when we took more points. One more interesting thing is that elapsed time, under equal conditions, equal in both cases.

Let's consider a function with a few extremes.

$$f(x) = e^{x-y} \cdot sin(8(x-y)) \cdot x$$

$$x \in [0,1]$$

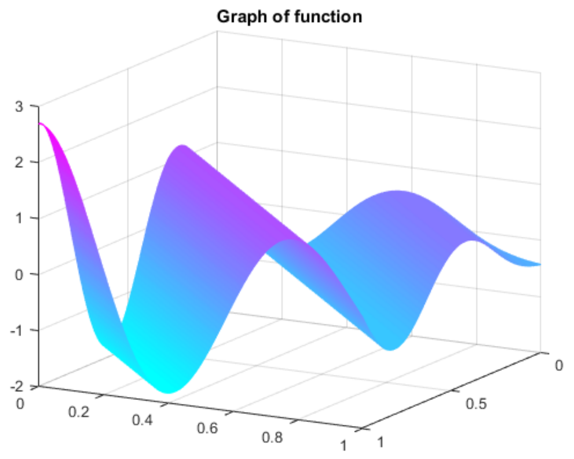


Figure 24. Graph of function

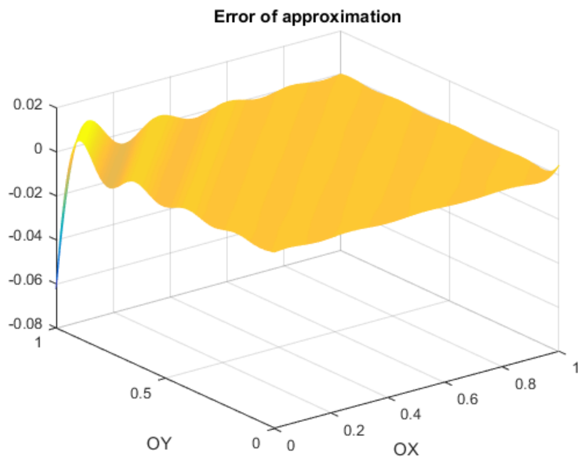


Figure 25. Error of approximation

- Number of neurons: 15
- Elapsed time: $17s$
- Number of approximation points: $100x100$

As we can see, elapsed time is't very big, that's why result is not very good. Below, there is an example of the same problem but with the different stopping conditions.

$$f(x) = e^{x-y} \cdot sin(8(x-y)) \cdot x$$

$$x \in [0,1]$$

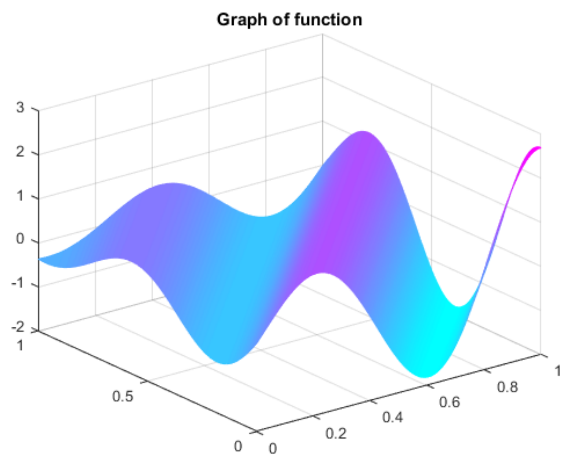


Figure 26. Graph of function

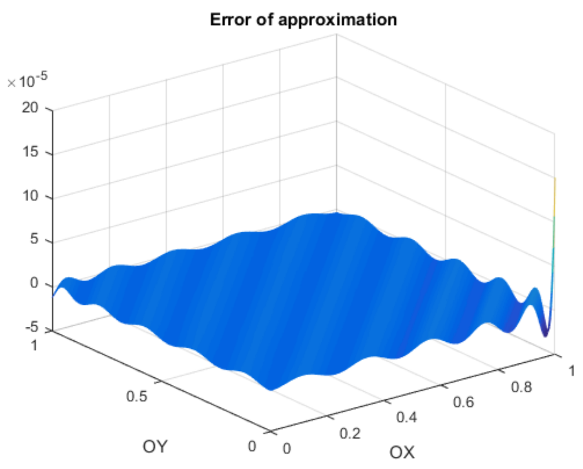


Figure 27. Error of approximation

- Number of neurons: 15
- Elapsed time: $47s$
- Number of approximation points: $100x100$

As we can see, the program run time increases to $47s$, and accuracy increases from 10^{-2} to 10^{-5}.

4.2. 1-dimensional integrals

$$f(x) = \frac{1}{1+x}$$

$$x \in [0,1]$$

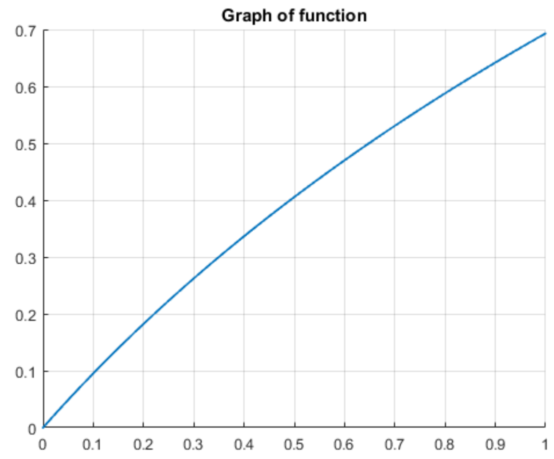


Figure 28. Graph of integral

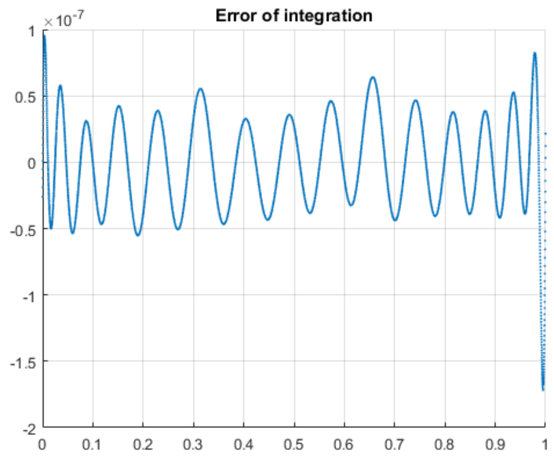


Figure 29. Error of integration

- Number of neurons: 15
- Elapsed time: $1s$
- Number of approximation points: 300

Very good results in short time.

$$f(x) = ln(cos(x))$$

$$x \in [0,1]$$

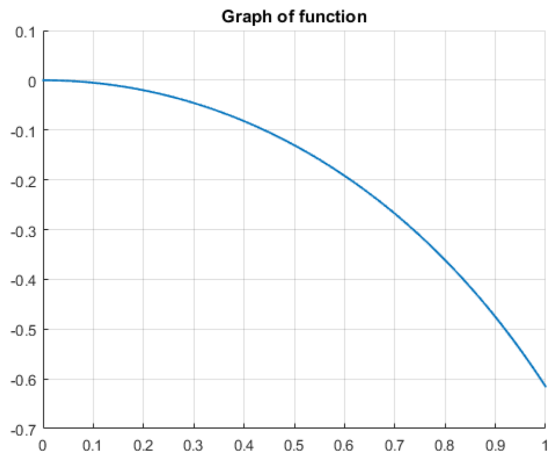


Figure 30. Graph of integral

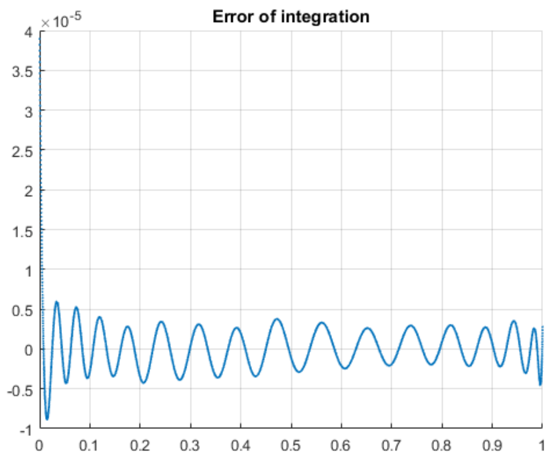


Figure 31. Error of integration

- Number of neurons: 15
- Elapsed time: $1s$
- Number of approximation points: 300

4.3. 2-dimensional integral

Let's consider next problem

$$\int_0^y dy' \int_0^x dx' f(x', y')$$

$$f(x', y' = e^{(x-y)\cdot sin(6(x-y))}$$

$$x \in [0, 1]$$

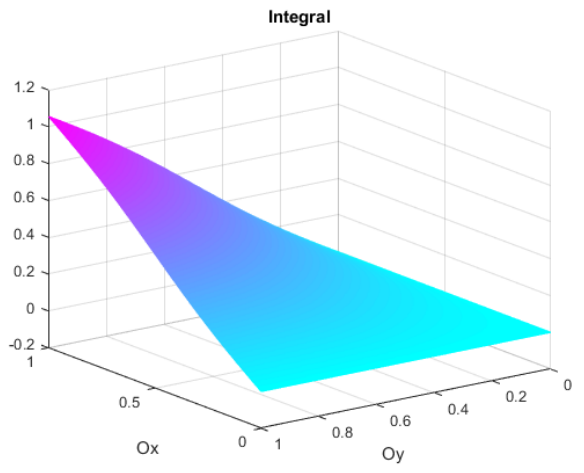


Figure 32. Graph of integral

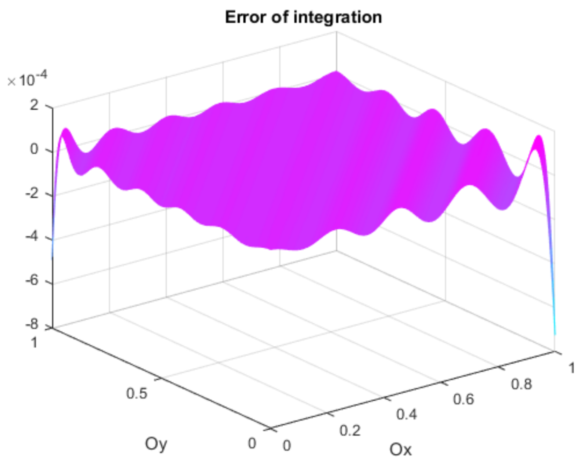


Figure 33. Error of integration

- Number of neurons: 15
- Elapsed time: $30s$
- Number of approximation points: 300

In fact, this problem should work with our scheme, but, really, it's very to transform our scheme for calculating such integrals.

5. Conclusion

During my stay at Dubna, I had done a lot of things. First of all, I met a lot of interesting people. Talking to them was a big pleasure for me. Secondly, I participated in different conferences and learnt a lot from them.

Our work, which was done with my supervisor Hovik Grigorian and our friend Alexander Ayriyan, was a good practice for me, and we are planning to continue our research in this field. At this moment, we created and researched algorithm, that could calculate n-dimensional integrals. During our research we faced with some problems, like problem of memory. This problems under consideration at the moment, and I believe, that we will manage them in the future.

I must note, that results of calculating 1- and 2- dimensional integrals, which are provided in this report, can compete with best methods, like Monte-Carlo method.

Also, I want to say thank you for JINR and Orginizing Comitee for this opportunity.

In addition, I want to say thank you for my supervisor, Hovik Grigorian, and Alexander Ayriyan. Both of them are professionals with a big experience. At the result, they became good friends for me.

6. References

- ”Approximation with Artificial Neural Networks, MSc thesis” by Balázs Csanád Csáji
- ”Neural Network” by R.Rojas, 1997