On Neural Network Approach for Numerical Integration of Single and Double Integrals

Gregory A. Shipunov[1]
supervisors: Oksana I. Streltsova,[1, 2] Yury L. Kalinovsky[1, 2]

[1]Dubna State University, 19 Universitetskaya St, Dubna, 141980, Russian Federation
[2]Joint Institute for Nuclear Research, 6 Joliot-Curie St, Dubna, 141980, Russian Federation

Table of Contents

Table of Contents

Applied Problem of Meson Properties Modeling Statement 1

The development of software implementation of the neural network approach to the numerical integration of single and double integrals was motivated by necessity to calculate value of multiple integrals in domain of modeling of physical processes for NICA collider. Numerical modeling of physical processes involves studying of the particle properties based on Bethe-Salpeter equation:

$$\Gamma(q,P) = -\frac{4}{3}\int \frac{d^4p}{(2\pi)^4} D(q-p)\gamma_\alpha S_1 \Gamma(q,P) S_2 \gamma_\alpha . \tag{1}$$

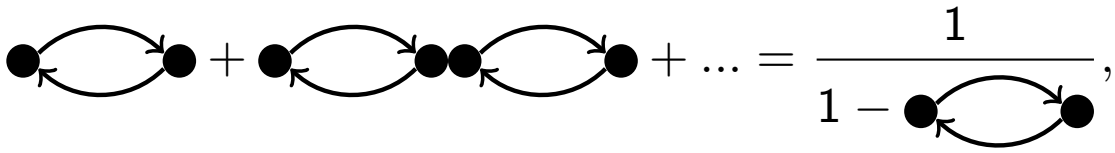
Equation (1) contains the interaction kernel $D(q-p)$ which describes the effective gluon interaction within a meson. We consider the rank -1 separable model

$$D(q-p) = D_0 F(q^2) F(p^2)\,, \tag{2}$$

where D_0 is the coupling constant and the function $F(p^2)$ is related to scalar part of Bethe - Salpeter vertex function. We employ $F(q^2)$ in the Gaussian form $F(p^2) = e^{-p^2/\lambda^2}$ with the parameter λ which characterizes the finite size of the meson.

Applied Problem of Meson Properties Modeling Statement 2

This separable Anzatz of the interaction kernel (2) allows to write the meson observables in the term of the polarization operator (the buble diagram):



(3)

where

$$\rightarrow \int_0^\infty \frac{dp}{2\pi^4} F(p^2) \frac{1}{[(p+q_1)^2+m_1^2][(p+q_2)^2+m_2^2]}. \tag{4}$$

To calculate these integrals we use the equations:

$$\frac{1}{(p+q_i)^2+m_i^2} = \int_0^\infty dt\{\exp(-t[(p+q_i)^2+m_i^2])\}, \tag{5}$$

$$F(p^2) = \int_0^\infty ds\{\exp(-sp^2)F(s)\}. \tag{6}$$

The case of more than two mesons is described using triple and quadruple integrals.

Applied Problem of Meson Properties Modeling Statement 3: Integral equation

This way the following integral equation is produced:

$$\int_0^1 d\alpha\{\alpha^a(1-\alpha)^b\}\int_0^\infty dt\{\frac{t^m}{(1+t)^n}F[z_0]\} \equiv I(a,b,m,n;F[z_0]), \tag{7}$$

$$F[z_0] = \exp(-2z_0),$$

$$z_0 = tD + \frac{t}{1+t}R^2,$$

$$D = \alpha_1(b_1^2P^2 + m_1^2) + \alpha_2(b_2^2P^2 + m_2^2),$$

$$R^2 = (\alpha_1^2b_1^2 + \alpha_2^2b_2^2 + 2\alpha_1\alpha_2b_1b_2)P^2,$$

$$b_1 = -\frac{m_1}{m_1+m_2}, b_2 = \frac{m_2}{m_1+m_2},$$

$$\alpha_1 = \alpha, \alpha_2 = 1-\alpha.$$

This integral is used in multiple calculations in the problem and its value describes the form of the mesons. The calculations require multiple (50 times and more) integrations of (7).

Applied Problem of Meson Properties Modeling Statement 4: Integrand shapes

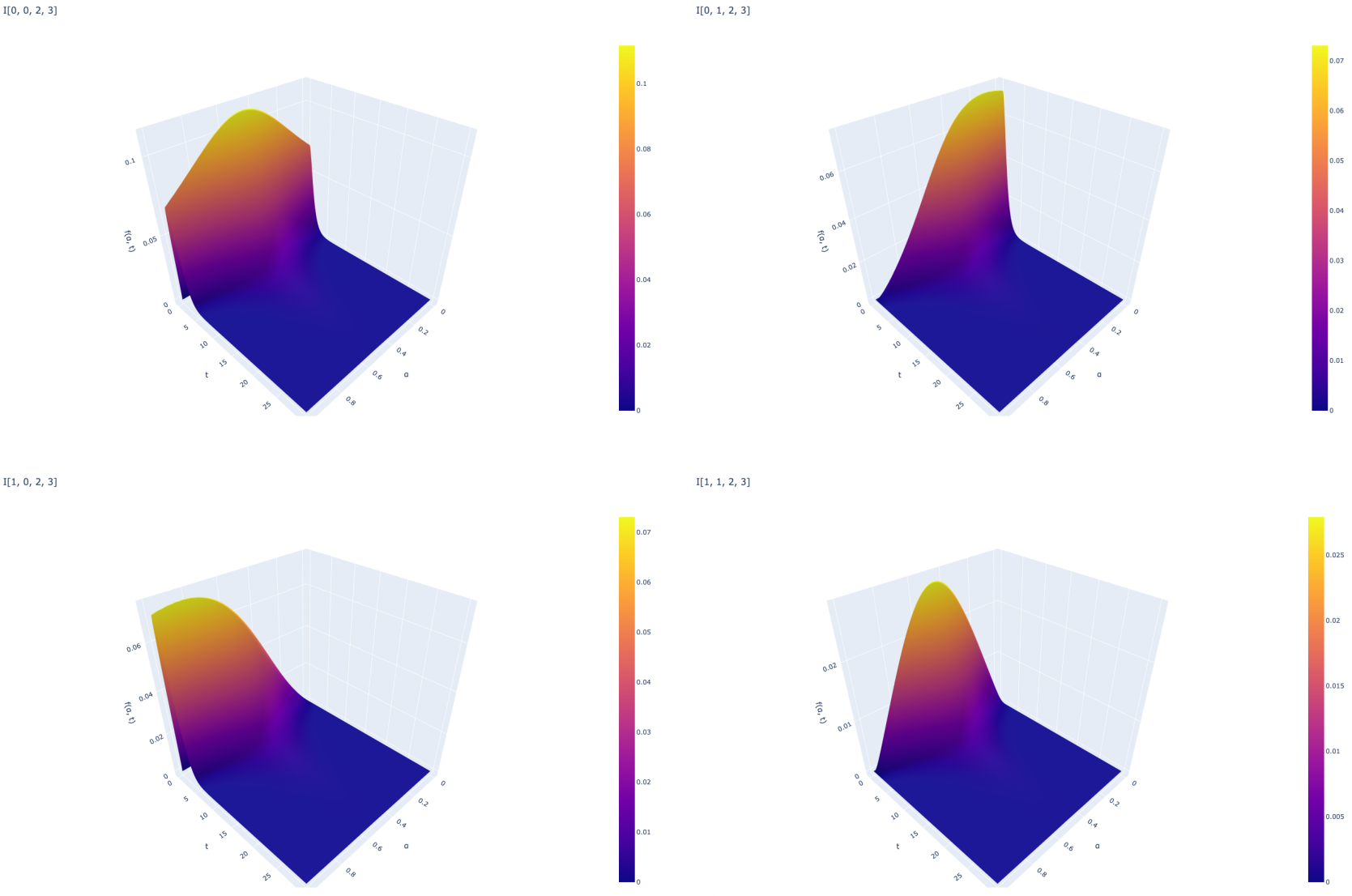


Figure 1: Plots of integrand of equation (7) with different a, b, m, n parameters

Mathematical foundation for the neural network approach 1

Let a continuous real function $f(\mathbf{x})$ be defined as $f : \mathbb{R}^n \to \mathbb{R}$. Let Ω be a compact subset of $\mathbb{R}^n$ and let G be a bounded convex subset of Ω. Let, also, $\mathbf{x}$ be a vector of n dimensions in Ω. Than

$$I(f) = \int_G d\mathbf{x} f(\mathbf{x}), \tag{8}$$

is a definite integral of f across set G.
Therefore, the problem is to get the $I(f)$ value calculated with use of neural network approach. The neural network model will be used to approximate the function $f(\mathbf{x})$[1].
Than this model will be used to calculate the approximate integral value:

$$\hat{I}(f) = \int_G d\mathbf{x} \hat{f}(\mathbf{x}). \tag{9}$$

[1] S. Lloyd, R. A. Irani, M. Ahmadi, Using neural networks for fast numerical integration and optimization, IEEE Access 8 (2020) 84519–84531.

Mathematical foundation for the neural network approach 2

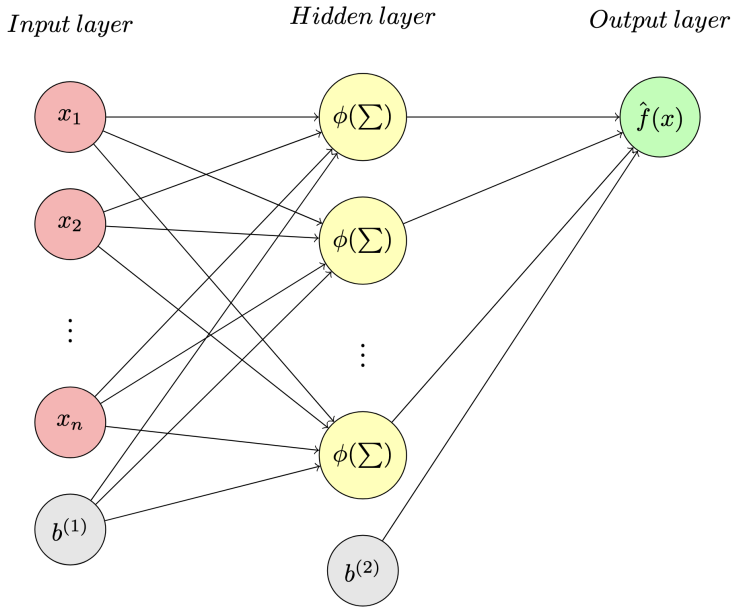


Figure 2: The MLP structure used in the neural network approach

The MLP structure will have logistic sigmoid as activation function on the hidden layer:

$$\phi(z) = \frac{1}{1 + \exp(-z)}. \tag{10}$$

Mathematical foundation for the neural network approach 3

With $\phi(z)$ defined, network structure's mathematical form is:

$$\hat{f}(x) = b^{(2)} + \sum_{j=1}^{k} w_j^{(2)} \phi(b_j^{(1)} + \sum_{i=1}^{n} w_{ji}^{(1)} x_i). \tag{11}$$

We can apply following substitution[2] here:

$$-Li_0(-\exp(z)) = \frac{1}{1+\exp(-z)} = \phi(z), \tag{12}$$

where $Li_0(u(z))$ is a Jonquière's function or the polylogarithm of order 0.
NB: the analytical form of polylogarithm of order n integral is quite trivial, which makes it easy to integrate it multiple times:

$$\int dz Li_s(-\exp[\kappa(z)]) = \frac{\partial \kappa(z)}{\partial z} Li_{s+1}(-\exp[\kappa(z)]). \tag{13}$$

[2] S. Lloyd, R. A. Irani, M. Ahmadi, Using neural networks for fast numerical integration and optimization, IEEE Access 8 (2020) 84519–84531.

Mathematical foundation for the neural network approach 4

Let number of dimensions $n = 1$. With substitution (12) equation (11) can be integrated over given boundaries $[\alpha, \beta]$ to produce the following numerical integration formulae for 1-dimensional case[3]:

$$\hat{I}(f) = \int_{\alpha}^{\beta} dx \left(b^{(2)} + \sum_{j=1}^{k} w_j^{(2)} \phi(b_j^{(1)} + w_{1j}^{(1)} x) \right) =$$

$$= b^{(2)}(\beta - \alpha) + \sum_{j=1}^{k} w_j^{(2)} \left((\beta - \alpha) + \frac{\Phi_j}{w_{1j}^{(1)}} \right), \tag{14}$$

$$\Phi_j = Li_1(-\exp[-b_j^{(1)} - w_{1j}^{(1)}\alpha]) - Li_1(-\exp[-b_j^{(1)} - w_{1j}^{(1)}\beta]). \tag{15}$$

Formulae (14-15) can be extrapolated to higher dimensions as seen below.

[3] S. Lloyd, R. A. Irani, M. Ahmadi, Using neural networks for fast numerical integration and optimization, IEEE Access 8 (2020) 84519–84531.

In general case (function of n variables) the substitution (12) reveals its full potential. For integrations over boundaries $[\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \ \ldots \ \times [\alpha_n, \beta_n]$ we get[4]:

$$\hat{I}(f) = b^{(2)} \prod_{i=1}^{n} (\beta_i - \alpha_i) + \sum_{j=1}^{k} w_j^{(2)} [\prod_{i=1}^{n} (\beta_i - \alpha_i) + \frac{\Phi_j}{\prod_{i=1}^{n} w_{ij}^{(1)}}], \tag{16}$$

$$\Phi_j = \sum_{r=1}^{2^n} \xi_r Li_n(-\exp[-b_j^{(1)} - \sum_{i=1}^{n} w_{ij}^{(1)} l_{i,r}]), \tag{17}$$

$$\xi_r = \prod_{d=1}^{n} (-1)^{[r/2^{n-d}]}, \tag{18}$$

$$l_{i,r} = \begin{cases} \alpha_i, & \text{if } \dfrac{r}{2^{n-d}} \text{ is even,} \\ \beta_i, & \text{if } \dfrac{r}{2^{n-d}} \text{ is not even.} \end{cases} \tag{19}$$

Thus we have got the formulae (16-19) describing numerical integration method.

[4] S. Lloyd, R. A. Irani, M. Ahmadi, Using neural networks for fast numerical integration and optimization, IEEE Access 8 (2020) 84519–84531.

Table of Contents

Neural Network Model & Training Process Description

The neural network model dedicated to approximate the $f(\mathbf{x})$ function was implemented using *Python* programming language and its *PyTorch* library and had following setting:

- Size of input layer **n** was equal to the number of function's dimensions.
- Size of hidden layer was $\mathbf{k} = 25$.

The training process took approximately **15 seconds** fo 1-dimensional functions and **30 seconds**[5] for 2-dimensional functions. The training was conducted under the following parameters:

- Optimizer **Adam** was used.
- Training was performed over **5000 epochs**.
- Dataset size was **40000 points**. 0.9 of the size was used for training and 0.1 for testing of the model.

The dataset structure was the **uniform grid** and the **standard uniform distribution** within given boundaries.

[5] All the training was performed with no parallelization on the following hardware: M1 Pro (8 cores), 16 GB RAM; and software: Jupyter Lab 4.2.5, Python 3.13.2, PyTorch 2.2.2.

Results 1: Genz Package

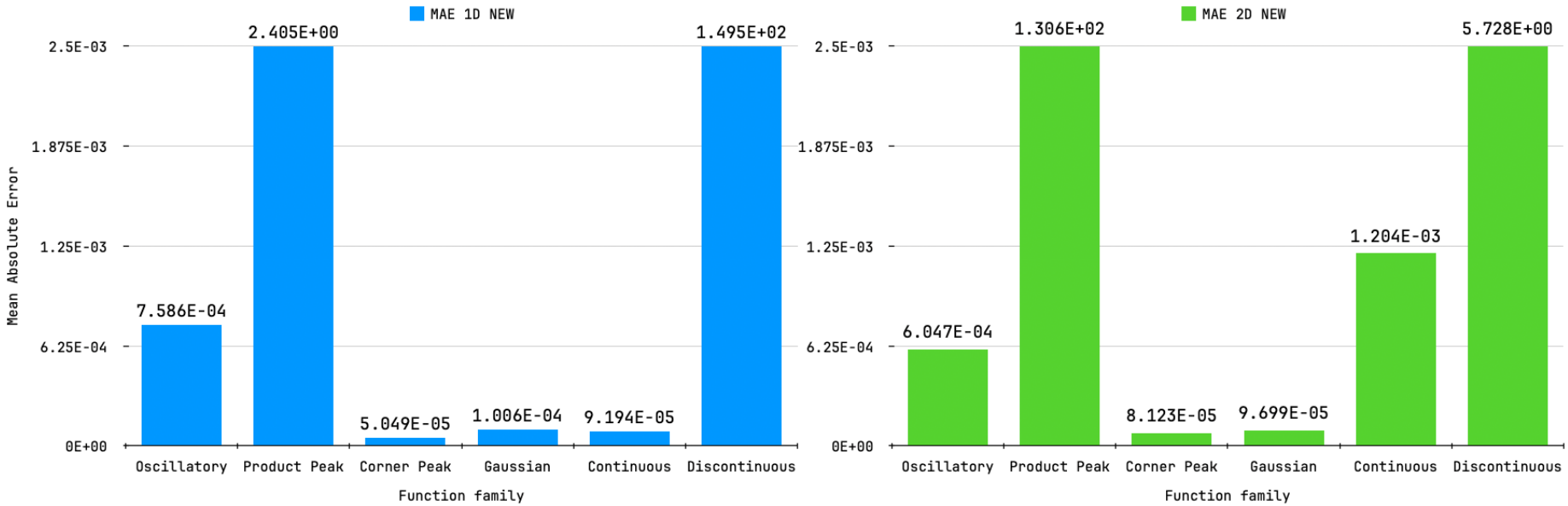


Figure 3: The Mean absolute error metrics for comparison of neural network numerical integration values and quad numerical integration values (Product Peak and Discontinuous bars are clipped for scale but their values are presented)

Figure 5 depicts the mean absolute errors of integration of Alan Genz package[6] 1D and 2D functions compared to results calculated using *scipy.integrate.quad* fucntion of Python programming language.

[6] A. Genz, Testing multidimensional integration routines, in: Proc. of international conference on Tools, methods and languages for scientific and engineering computation, 1984, pp. 81–94.

Results 2: Hidden layer size

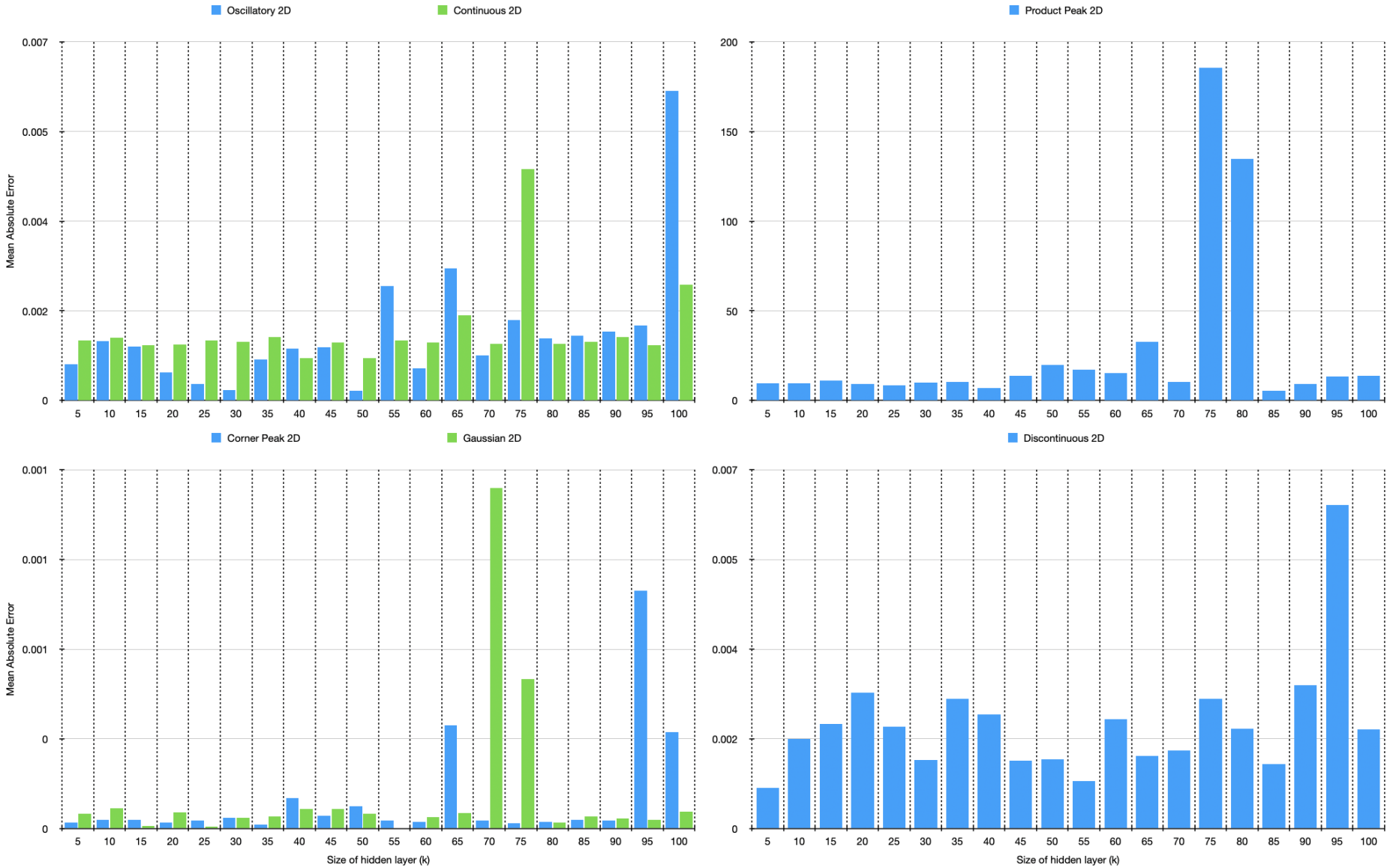


Figure 4: The Mean absolute error metrics relative to the size of hidden layer (k) – **[10, 100]**

Results 3: Distribution type

Figure 5: The Mean absolute error metrics relative to the type of dataset distribution (**UG** - Uniform Grid and **SUD** - Standard Uniform Distribution) with different dataset sizes – **[100, 40000]**

Results 4: Equation (7)

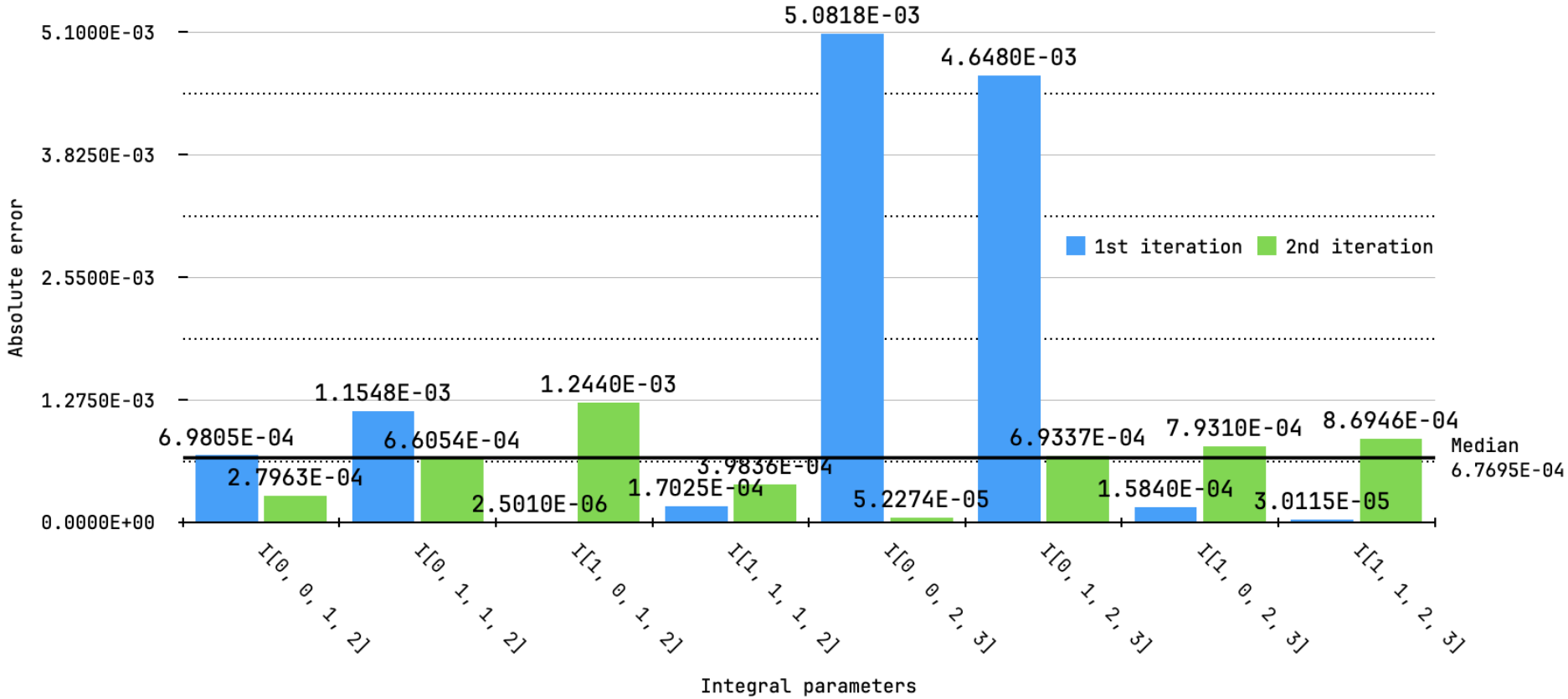


Figure 6: The absolute errors between numerical integral values calculated using neural network approach and FORTRAN function qdag

Table of Contents

The Conclusion: Benefits of using neural network approach. Future development plans

The usage of the neural network approach to the numerical integration has following benefits:

- The integration of similar function across different boundaries **time efficiency**.
- The integration of **functions of high-dimensions**.

Currently the software implementation of the approach is a library for **Python** language named **Skuld**. And there is still much to do in this field, especially:

- **Improve accuracy** of integration by adjusting data distribution and optimization algorithms.
- Develop **the general fine-tuning routine** for the particular integrand classes.
- Implement **higher dimensions** numerical integration functionality.
- Consider **other approaches** to use neural networks for numerical integration.

Acknowledgments

- To the supervisors **Oksana Streltsova** and **Yury Kalinovsky** for their scientific guidance.
- To **Tatevik Bezhanyan** and **Sara Shadmehri** for their advices.

Thank You for Your Attention!

Apendix A: Genz's testing functions families 1

Oscillatory (1)

$$f(\mathbf{x}) = \cos(2\pi u_1 + \sum_{i=1}^{n} c_i x_i).$$

Product Peak (2)

$$f(\mathbf{x}) = \prod_{i=1}^{n} (c_i^{-2} + (x_i - u_i)^2)^{-1}.$$

Corner Peak (3)

$$f(\mathbf{x}) = (1 + \sum_{i=1}^{n} c_i x_i)^{-(n+1)}.$$

Gaussian (4)

$$f(\mathbf{x}) = \exp(-\sum_{i=1}^{n} c_i^2 (x_i - u_i)^2).$$

Continuous (5)

$$f(\mathbf{x}) = \exp(-\sum_{i=1}^{n} c_i |x_i - u_i|).$$

Discontinuous (6)

$$f(\mathbf{x}) = \begin{cases} 0 & \text{if any of } x_i > u_i, \\ \exp(\sum_{i=1}^{n} c_i x_i) & \text{alternatively.} \end{cases}$$

Apendix A: Genz's testing functions families 2

The n value is the number of dimensions of the integrand. This functions contain n-vectors $\mathbf{u}$ and $\mathbf{c}$ in their definition.

1. u_i is a random value in [0, 1].
2. $\mathbf{c} = \left(\frac{h}{n^j \sum_{i=1}^{n} c_i'} \right) \mathbf{c}'$, $\mathbf{c}'$ is and n-vector, whith c_i' is a random value in [0, 1].

The h and j are **difficulty** parameters. In the case of this study the last parameter was $j = 1$ and h parameter is equal to $(100, 150, 600, 150, 100, 16)$ accordingly for each of the function families.