On the Calculation of Feynman Integrals via Neural Network Approximation of Antiderivatives

Shipunov G. A.[1,2], Streltsova O. I.[1, 2], Kalinovsky Yu. L.[1, 2]

[1]Meshcheryakov Laboratory of Information Technologies, Joint Institute for Nuclear Research, 6 Joliot-Curie St., Dubna, 141980, Russia
[2]Dubna Branch of Lomonosov Moscow State University, 12 Leningradskaya St., Dubna, 141980, Russia



MATHEMATICAL MODELING AND COMPUTATIONAL PHYSICS 2026

June 25, 2026

Applied Problem: Modelling Meson Properties 1

The development of a neural network-based software for numerical integration was motivated by the need to evaluate two-, three-, and four-dimensional integrals arising in the numerical simulation of physical processes at the NICA collider. This simulation involves studying particle properties within the framework of the Bethe–Salpeter equation[1][2][3]:

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

Expression (1) contains the interaction kernel $D(q-p)$, which describes the effective quark–quark interaction; γ_α denotes the Dirac matrices, and S_1, S_2 are the quark propagators. We adopt a separable model of the form:

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

where D_0 is the coupling constant and the function $F(p^2)$ belongs to the scalar part of the vertex function equation for $\Gamma(p,P)$. We choose $F(p^2)$ in Gaussian form: $F(p^2) = e^{-p^2/\lambda^2}$, where the parameter λ characterises the finite size of the meson.

[1] Blaschke, Grigorian and Kalinovsky, Meson form-factor scheme for the chiral lagrangian approach to J/ψ breakup cross sections motivated by a relativistic quark model, 2012

[2] Costa, Ruivo and Kalinovsky, Pseudoscalar neutral mesons in hot and dense matter, 2003

[3] Schmidt, Blaschke and Kalinovsky, Scalar-pseudoscalar meson masses in nonlocal effective QCD at finite temperature, 1994

Applied Problem: Modelling Meson Properties 2

The choice of interaction kernel (2) allows us to express meson observables (masses, decay constants) in terms of the polarisation operator:



$$= \frac{1}{1 - \quad}, \tag{3}$$

where



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

To evaluate these integrals, we make use of the following representations:

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

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

The cases of interactions, decays, and scatterings are described by 3-d and 4-d integrals.

Applied Problem: Modelling Meson Properties 3 — Integral Equation

$$\int_0^1 d\alpha_1 \alpha_1^a \int_0^{1-\alpha_1} d\alpha_2 \alpha_2^b \int_0^\infty dt \frac{t^m}{(1+t)^n} F[z_0] \equiv I(a,b,m,n;F[z_0]) \tag{7}$$

$$F[z_0] = \exp[-z_0],$$

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

$$D = (\alpha_1\alpha_2)P^2 + P_1^2\alpha_2(1-\alpha_1-\alpha_2) + P_2^2\alpha_1(1-\alpha_1-\alpha_2)$$
$$+\alpha_1 m_1^2 + \alpha_2 m_2^2 + (1-\alpha_1-\alpha_2)m_3^2.$$

$$R^2 = \alpha_1^2 P_2^2 + \alpha_2^2 P_1^2 - (\alpha_1\alpha_2)(P^2 - P_1^2 - P_2^2),$$

This expression arises in various computations within the problem described above, and its values characterise the meson form factors. The computations require repeated evaluation (50 or more times) of the integral (7), as well as other integrals of a similar structure.
We imply the usage of neural networks for numerical calculation of Feynman multidimensional integrals such as 7 and thus a Python neural network numerical integration library Skuld is curently under development.

1. Approximation of the Integrand 1[4]

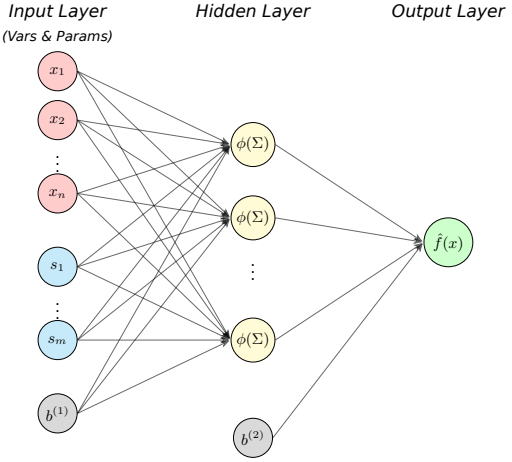


Figure 1: Neural network $\mathcal{N}$ approximating the integrand $f(x)$

Let $f(\mathbf{x}, \mathbf{s})$ be a continuous function of real variables of the form $f : \Re^{n+m} \to \Re$. Let $\Omega \subset \Re^n$, let S^m be a space of descrete parameters of the function f and let G be a convex subset of Ω. Furthermore, let $\mathbf{x} \in \Omega^n$ denote the vector of integration variables and $\mathbf{s} \in S^m$ the vector of function parameters. Then

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

is the definite integral of f over the domain G. The problem thus reduces to evaluating $I(f)$ by means of a neural network approach. The neural network model is used to approximate $f(\mathbf{x})$, after which this model is employed to compute an approximate value of the integral:

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

[4] Lloyd, Irani and Ahmadi, Using neural networks for fast numerical integration and optimization, IEEE Access 2020

1. Approximation of the Integrand 2

In the general case (a function of n variables and m parameters), the analytical form of the integral of the neural network model over the domain $[\alpha_1, \beta_1] \times [\alpha_2, \beta_2] \times \cdots \times [\alpha_n, \beta_n]$ can be written as:

$$\hat{I}(f, \boldsymbol{s}) = b^{(2)} \prod_{i=1}^{n} (\beta_i - \alpha_i) + \sum_{j=1}^{k} w_j^{(2)} \left[\prod_{i=1}^{n} (\beta_i - \alpha_i) + \frac{\Phi_j}{\prod_{i=1}^{n} w_{ij}^{(1)}} \right], \tag{10}$$

$$\Phi_j = \sum_{r=1}^{2^n} \xi_r \, Li_n \left(-\exp\left[-\tilde{b}_j^{(1)} - \sum_{i=1}^{n} w_{ij}^{(1)} l_{i,r} \right] \right), \tag{11}$$

$$\tilde{b}_j^{(1)} = b_j^{(1)} + \sum_{q=1}^{p} w_{jq}^{(1,s)} \, s_q, \tag{12}$$

$$\xi_r = \prod_{d=1}^{n} (-1)^{\left\lceil r/2^{n-d} \right\rceil}, \tag{13}$$

$$l_{i,r} = \begin{cases} \alpha_i, & \text{if } \left\lfloor \frac{r}{2^{n-d}} \right\rfloor \text{ is even,} \\ \beta_i, & \text{if } \left\lfloor \frac{r}{2^{n-d}} \right\rfloor \text{ is odd.} \end{cases} \tag{14}$$

Here $w_{ij}^{(1)}$ are the weights corresponding to the variables, $w_{jq}^{(1,s)}$ are the weights corresponding to the function parameters, and $b^{(1)}, b^{(2)}, \mathbf{w}^{(2)}$ are the neural network parameters. This defines the numerical integration method based on approximation of the integrand.

2. Approximation of the Antiderivative of the Integrand 1[5]

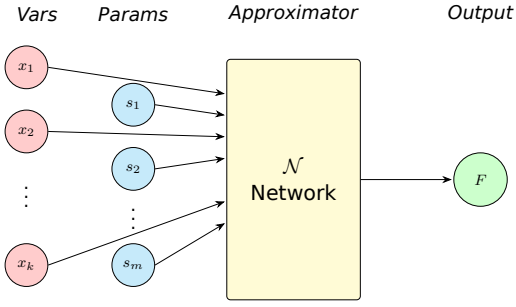


Figure 2: Neural network $\mathcal{N}$ approximating the antiderivative F

We consider a parametric integral of the form:

$$I(s_1, \ldots, s_m) = \int_0^1 \cdots \int_0^1 dx_1 \cdots dx_k \; f(\mathbf{s}, \mathbf{x}). \tag{15}$$

The neural network $\mathcal{N}(\mathbf{s}; \mathbf{x})$ is trained to approximate the antiderivative F satisfying:

$$\frac{\partial^k F}{\partial x_1 \cdots \partial x_k} = f(\mathbf{s}, \mathbf{x}). \tag{16}$$

The integral is then evaluated analytically via the boundary values of the network:

$$I(\mathbf{s}) = \sum_{\substack{x_1, \ldots, x_k \\ \in \{0,1\}}} (-1)^{k - \sum_i x_i} \mathcal{N}(\mathbf{s}, \mathbf{x}). \tag{17}$$

Key distinction from Lloyd et al.: the network approximates not the integrand itself, but its antiderivative.

[5] D. Maître, R. Santos-Mateos, Multi-variable integration with a neural network, JHEP 2023

Loss function and activation

The network is trained to minimise the MSE between the k-th mixed partial derivative of the output and the value of the integrand, by adjusting the weights $\mathbf{W}$:

$$\mathcal{L}(\mathbf{W}) = \mathrm{MSE}\left(f(\mathbf{s}, \mathbf{x}),\ \frac{\partial^k \mathcal{N}(\mathbf{s}, \mathbf{x})}{\partial x_1 \cdots \partial x_k} \right). \tag{18}$$

To prevent the network's derivatives from vanishing, a family of activation functions related by $f_n' = f_{n-1}$ must be employed. For the sigmoid ($f_k = \sigma$):

$$f_0(x) = \sigma(x) = \tfrac{1}{1+e^{-x}}, \tag{19}$$

$$f_1(x) = \log(1 + e^x), \tag{20}$$

$$f_n(x) = -\mathrm{Li}_n(-e^x). \tag{21}$$

Layer derivatives are computed recursively. For example, in case of two variables:

$$\begin{aligned} \frac{\partial^2 a_i^{(l)}}{\partial x_1 \partial x_2} = \phi'' \left(\sum_j w_{ij} \frac{\partial a_j^{(l-1)}}{\partial x_1} \right) \left(\sum_j w_{ij} \frac{\partial a_j^{(l-1)}}{\partial x_2} \right) + \\ + \phi' \left(\sum_j w_{ij} \frac{\partial^2 a_j^{(l-1)}}{\partial x_1 \partial x_2} \right). \end{aligned} \tag{22}$$

Preprocessing: normalisation and the Korobov[6] transformation

The integrand is normalised by its value at the centre of the hypercube to stabilise training:

$$\tilde{f}(\mathbf{s}, \mathbf{x}) = \frac{f(\mathbf{s}, \mathbf{x})}{f\left(\mathbf{s}, \frac{1}{2}, \ldots, \frac{1}{2}\right)}. \tag{23}$$

The Korobov transformation is applied to each variable x_i: the substitution $x_i \mapsto t_i$ with weight function $w(t) = 6t(1-t)$:

$$x(t) = t^2(3-2t), \quad \int_0^1 dx\, g(x) = \int_0^1 dt\, w(t)\, g(x(t)). \tag{24}$$

This forces the integrand to vanish at the domain boundaries, thereby improving accuracy.

Initialisation and training

Weights are initialised using the Xavier scheme with a *gain* coefficient depending on the activation function.
Training is performed with the Adam optimiser.
This defines the numerical integration method based on approximation of the antiderivative of the integrand.

[6] Korobov N. M. Number-Theoretic Methods of Approximate Analysis. GIFL, 1963

Neural Network Architectures Considered for Antiderivative Approximation

Model	Activation	Key idea	Reference
MLP	sigmoid	Basic perceptron	Maître et al., 2023
SIREN	$\sin(\omega_0 x)$	Periodic activations	Sitzmann et al., 2020
WIRE	Gabor wavelet	Wavelet activations	Saragadam et al., 2023
KAN	B-splines	Learnable edge functions	Liu et al., 2024
BrokNet	arbitrary	Parameter-based routing between sub-networks	Shipunov et al., 2026

Table 1: Architectures considered as the approximator $\mathcal{N}$

All models are used to approximate the antiderivative F of the integrand f. Architectural requirements:

- $\mathcal{N}$ must be k times differentiable with respect to the variables $x_1, \ldots, x_k$.
- The derivatives $\partial^k \mathcal{N} / \partial x_1 \cdots \partial x_k$ must not vanish during training.
- The architecture must support efficient computation of mixed partial derivatives.

Preprocessing of the Integral (7)

Intial integrand:

$$\alpha_1^a \alpha_2^b \frac{t^m}{(1+t)^n}\, e^{-\left(t\,D + \frac{t}{1+t}\,R_2\right)}.$$

Integration domain (simplex $\to$ unit square):

$$\alpha_1 = x_1, \quad \alpha_2 = (1-x_1)\,x_2, \quad \alpha_3 = 1-\alpha_1-\alpha_2, \quad J_\alpha = (1-x_1).$$

Korobov transformation (removal of corner singularities, $x_i \in [0,1]$):

$$x_i = \varphi(u_i) \;=\; u_i^2(3-2u_i), \qquad \frac{dx_i}{du_i} = 6\,u_i(1-u_i).$$

$$\varphi'(0) = \varphi'(1) = 0 \quad \Rightarrow \quad \text{vanishing derivatives at boundaries.}$$

Substitution for the variable $t \in [0,+\infty)$:

$$t = e^{Lu_3} - 1, \qquad dt = L\,e^{Lu_3}\,du_3, \quad u_3 \in [0,1].$$

$$\frac{t}{1+t} = 1 - e^{-Lu_3} \;\longrightarrow\; [0,1).$$

Resulting integrand form on $[0,1]^3$:

$$\tilde{f}(u_1,u_2,u_3) = \alpha_1^a\,\alpha_2^b\,\frac{t^m}{(1+t)^n}\, e^{-\left(t\,D + \frac{t}{1+t}\,R_2\right)} \cdot J_\alpha \cdot 6u_1(1-u_1) \cdot 6u_2(1-u_2) \cdot L\,e^{Lu_3}.$$

Employed Accuracy Metrics

Absolute error:

$$E_{abs} = |I_{num} - I_{nni}|, \tag{25}$$

where I_{num} is the reference value of the integral obtained by the Quasi-Monte Carlo method[7] of the pySecDec library[8], and I_{nni} is the integral value computed by the neural network-based method using Skuld library. NB: The error ε of the QMC method used to obtain the reference values was 10^{-9} and 10^{-10} for different parameter sets.
Relative error:

$$E_{rel} = \frac{E_{abs}}{|I_{num}|}. \tag{26}$$

Number of matching significant digits:

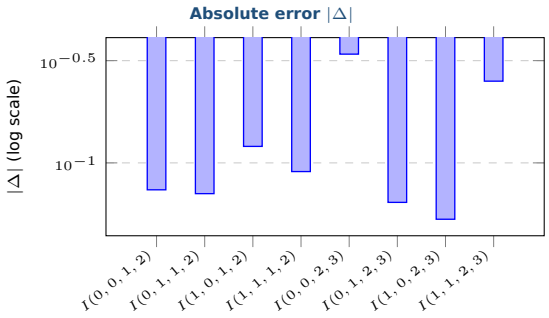
$$C_d = \max(0, \lfloor -\log_{10}(E_{abs} + \delta) \rfloor), \tag{27}$$

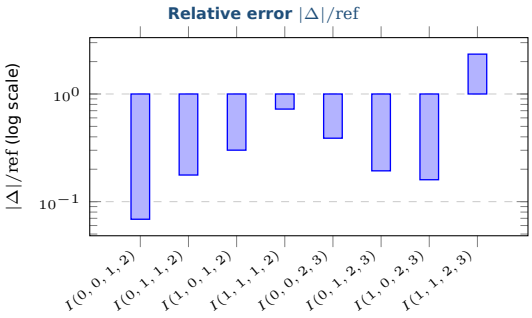
where $\delta \sim 10^{-30}$ is a regularisation term to handle the case of a zero absolute error.

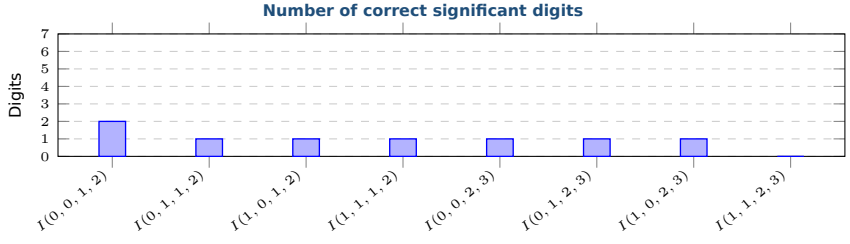
[7] Li, Zhao, et al. Efficient numerical evaluation of Feynman integrals. Chinese Physics C 40.3 2016

[8] Borowka, Sophia, et al. pySecDec: a toolbox for the numerical evaluation of multi-scale integrals. Computer Physics Communications 222 2018

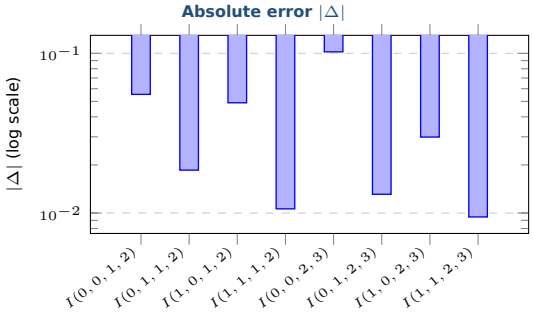
MLP: Results — Integrand Approximation Method

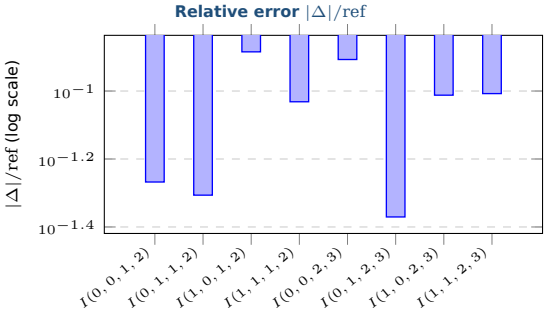


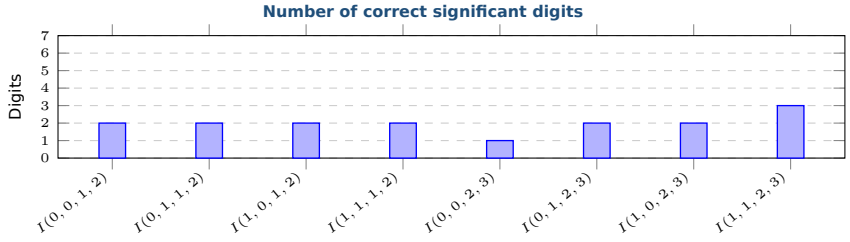




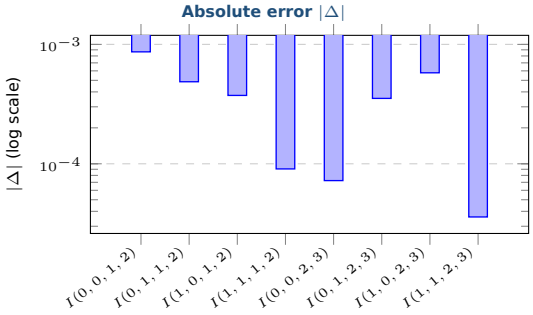
MLP: Results — Antiderivative Approximation Method

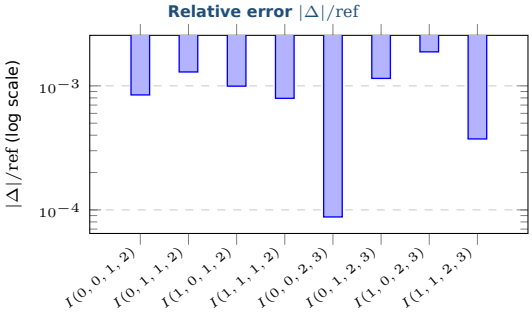


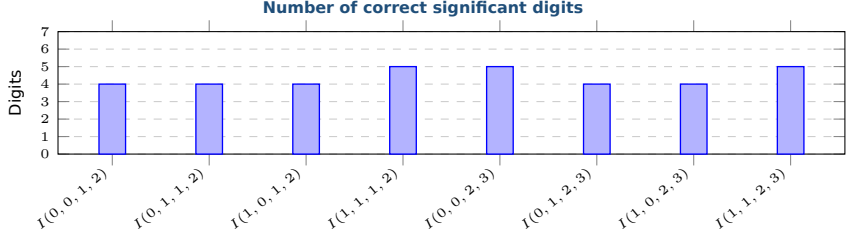




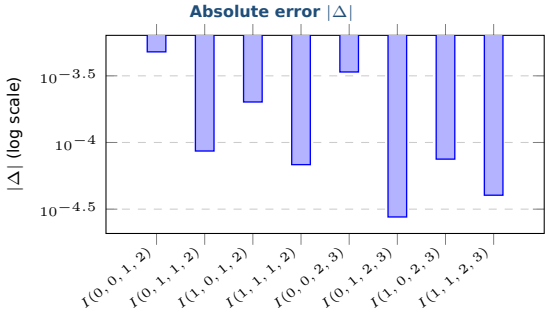
SIREN: Results — Antiderivative Approximation Method

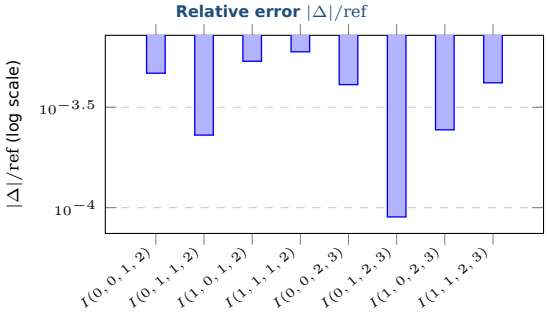


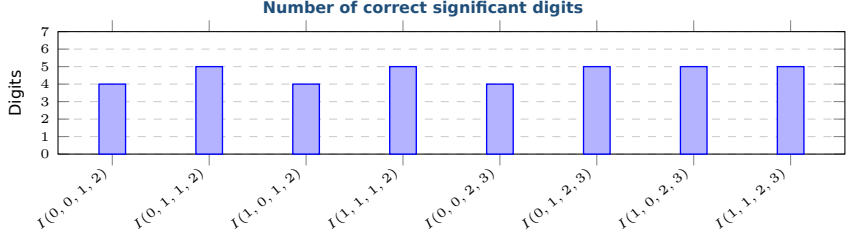




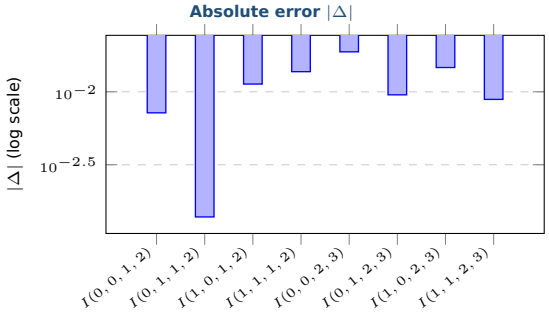
WIRE: Results — Antiderivative Approximation Method

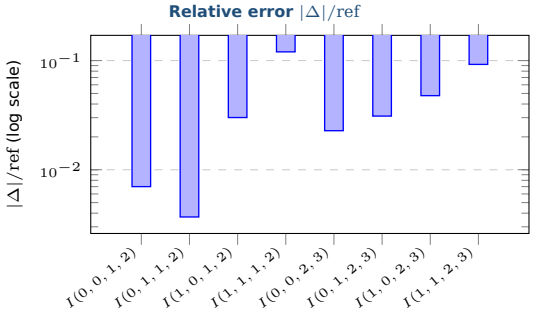


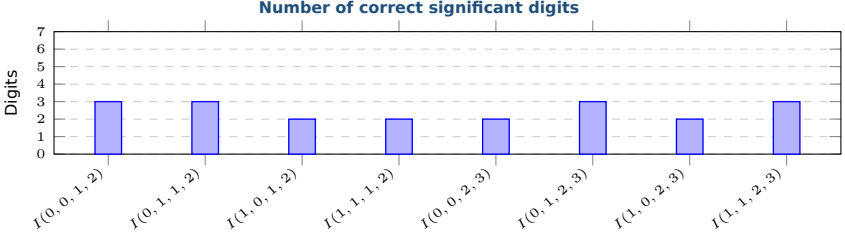




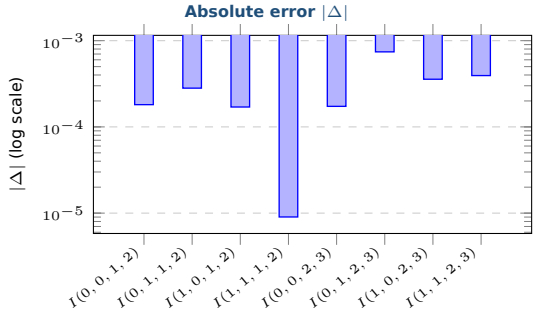
KAN: Results — Antiderivative Approximation Method

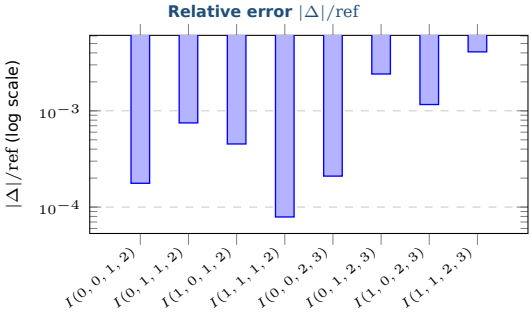


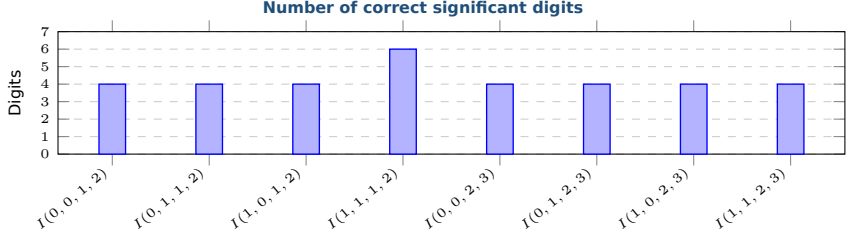




BrokNet: Results — Antiderivative Approximation Method







Summary Comparison of Methods

(a, b, m, n)	Lloyd	Maître	SIREN	WIRE	KAN	BrokNet
(0,0,1,2)	2	2	4	4	3	4
(0,1,1,2)	1	2	4	5	3	4
(1,0,1,2)	1	2	4	4	2	4
(1,1,1,2)	1	2	5	5	2	6
(0,0,2,3)	1	1	5	4	2	4
(0,1,2,3)	1	2	4	5	3	4
(1,0,2,3)	1	2	4	5	2	4
(1,1,2,3)	0	3	5	5	3	4

Time (s)	Lloyd	Maître	SIREN	WIRE	KAN	BrokNet
Time (s)	42.2	296.2	573.2	1938.4	6586.1	1979.6

BrokNet achieves the highest accuracy (up to **6** significant digits) at a moderate computational cost. **WIRE** and **SIREN** consistently deliver **4–5** digits.

Thank you for your attention!

Questons & Discussion

Contact: **shipunov@jinr.ru**.



MATHEMATICAL MODELING AND COMPUTATIONAL PHYSICS 2026

BACKUP MLP: Results — Integrand Approximation Method

(a,b,m,n)	MLP-NNI	pySecDec (ref)	$\lvert\Delta\rvert$	$\lvert\Delta\rvert/\mathrm{ref}$	Digits
(0,0,1,2)	1.1499964×10^{0}	1.0761455×10^{0}	7.385×10^{-2}	6.863×10^{-2}	2
(0,1,1,2)	3.2954237×10^{-1}	4.0026054×10^{-1}	7.072×10^{-2}	1.767×10^{-1}	1
(1,0,1,2)	2.7988917×10^{-1}	4.0026054×10^{-1}	1.204×10^{-1}	3.007×10^{-1}	1
(1,1,1,2)	2.1593356×10^{-1}	1.2524925×10^{-1}	9.068×10^{-2}	7.240×10^{-1}	1
(0,0,2,3)	5.3567803×10^{-1}	8.7599864×10^{-1}	3.403×10^{-1}	3.885×10^{-1}	1
(0,1,2,3)	3.9569595×10^{-1}	3.3159596×10^{-1}	6.410×10^{-2}	1.933×10^{-1}	1
(1,0,2,3)	3.8460150×10^{-1}	3.3159596×10^{-1}	5.301×10^{-2}	1.598×10^{-1}	1
(1,1,2,3)	3.5799837×10^{-1}	1.0701701×10^{-1}	2.510×10^{-1}	2.345×10^{0}	0

Integrals reproduced to **0–2 significant digits**.
Maximum relative error $\approx 2.3 \times 10^{0}$.

BACKUP MLP: Results — Antiderivative Approximation Method

(a,b,m,n)	MLP-NNI	pySecDec (ref)	$\lvert\Delta\rvert$	$\lvert\Delta\rvert/\mathrm{ref}$	Digits
(0,0,1,2)	1.079635×10^{0}	1.024334×10^{0}	5.530×10^{-2}	5.399×10^{-2}	2
(0,1,1,2)	3.943691×10^{-1}	3.758095×10^{-1}	1.856×10^{-2}	4.939×10^{-2}	2
(1,0,1,2)	4.248021×10^{-1}	3.758095×10^{-1}	4.899×10^{-2}	1.304×10^{-1}	2
(1,1,1,2)	1.248801×10^{-1}	1.142569×10^{-1}	1.062×10^{-2}	9.298×10^{-2}	2
(0,0,2,3)	9.268410×10^{-1}	8.247579×10^{-1}	1.021×10^{-1}	1.238×10^{-1}	1
(0,1,2,3)	3.205138×10^{-1}	3.074143×10^{-1}	1.310×10^{-2}	4.261×10^{-2}	2
(1,0,2,3)	3.373044×10^{-1}	3.074143×10^{-1}	2.989×10^{-2}	9.723×10^{-2}	2
(1,1,2,3)	1.055942×10^{-1}	9.614571×10^{-2}	9.448×10^{-3}	9.827×10^{-2}	3

All integrals reproduced to **1–3 significant digits**.
Maximum relative error $\approx 1.3\times10^{-1}$.

BACKUP SIREN: Results — Antiderivative Approximation Method

(a,b,m,n)	Siren-NNI	pySecDec	$\lvert\Delta\rvert$	$\lvert\Delta\rvert/\mathrm{ref}$	Digits
(0,0,1,2)	1.025200×10^{0}	1.024334×10^{0}	8.657×10^{-4}	8.451×10^{-4}	4
(0,1,1,2)	3.762960×10^{-1}	3.758095×10^{-1}	4.864×10^{-4}	1.294×10^{-3}	4
(1,0,1,2)	3.761833×10^{-1}	3.758095×10^{-1}	3.737×10^{-4}	9.944×10^{-4}	4
(1,1,1,2)	1.141663×10^{-1}	1.142569×10^{-1}	9.061×10^{-5}	7.930×10^{-4}	5
(0,0,2,3)	8.246855×10^{-1}	8.247579×10^{-1}	7.234×10^{-5}	8.771×10^{-5}	5
(0,1,2,3)	3.077671×10^{-1}	3.074143×10^{-1}	3.529×10^{-4}	1.148×10^{-3}	4
(1,0,2,3)	3.079923×10^{-1}	3.074143×10^{-1}	5.780×10^{-4}	1.880×10^{-3}	4
(1,1,2,3)	9.610982×10^{-2}	9.614571×10^{-2}	3.589×10^{-5}	3.733×10^{-4}	5

All integrals reproduced to **4–5 significant digits**.
Maximum relative error $\approx 2\times10^{-3}$.

BACKUP WIRE: Results — Antiderivative Approximation Method

(a,b,m,n)	WIRE	pySecDec	$\lvert\Delta\rvert$	$\lvert\Delta\rvert/\mathrm{ref}$	Digits
(0,0,1,2)	1.024812×10^{0}	1.024334×10^{0}	4.784×10^{-4}	4.670×10^{-4}	4
(0,1,1,2)	3.758959×10^{-1}	3.758095×10^{-1}	8.631×10^{-5}	2.297×10^{-4}	5
(1,0,1,2)	3.760109×10^{-1}	3.758095×10^{-1}	2.013×10^{-4}	5.357×10^{-4}	4
(1,1,1,2)	1.143251×10^{-1}	1.142569×10^{-1}	6.820×10^{-5}	5.969×10^{-4}	5
(0,0,2,3)	8.250958×10^{-1}	8.247579×10^{-1}	3.380×10^{-4}	4.098×10^{-4}	4
(0,1,2,3)	3.074419×10^{-1}	3.074143×10^{-1}	2.763×10^{-5}	8.989×10^{-5}	5
(1,0,2,3)	3.074893×10^{-1}	3.074143×10^{-1}	7.504×10^{-5}	2.441×10^{-4}	5
(1,1,2,3)	9.618594×10^{-2}	9.614571×10^{-2}	4.023×10^{-5}	4.184×10^{-4}	5

All integrals reproduced to **4–5 significant digits**.
Maximum relative error $\approx 6\times10^{-4}$.

BACKUP KAN: Results — Antiderivative Approximation Method

(a,b,m,n)	KAN	pySecDec	$\lvert\Delta\rvert$	$\lvert\Delta\rvert/\mathrm{ref}$	Digits
(0,0,1,2)	1.031509×10^{0}	1.024334×10^{0}	7.175×10^{-3}	7.004×10^{-3}	3
(0,1,1,2)	3.771969×10^{-1}	3.758095×10^{-1}	1.387×10^{-3}	3.692×10^{-3}	3
(1,0,1,2)	3.645072×10^{-1}	3.758095×10^{-1}	1.130×10^{-2}	3.007×10^{-2}	2
(1,1,1,2)	1.005201×10^{-1}	1.142569×10^{-1}	1.374×10^{-2}	1.202×10^{-1}	2
(0,0,2,3)	8.059727×10^{-1}	8.247579×10^{-1}	1.879×10^{-2}	2.278×10^{-2}	2
(0,1,2,3)	2.978824×10^{-1}	3.074143×10^{-1}	9.532×10^{-3}	3.101×10^{-2}	3
(1,0,2,3)	2.927453×10^{-1}	3.074143×10^{-1}	1.467×10^{-2}	4.772×10^{-2}	2
(1,1,2,3)	8.727675×10^{-2}	9.614571×10^{-2}	8.869×10^{-3}	9.224×10^{-2}	3

All integrals reproduced to **2–3 significant digits**.
Maximum relative error $\approx 1.2\times10^{-1}$.

BACKUP BrokNet: Results — Antiderivative Approximation Method

(a,b,m,n)	BrokNet	pySecDec	$\lvert\Delta\rvert$	$\lvert\Delta\rvert/\mathrm{ref}$	Digits
(0,0,1,2)	1.024153×10^{0}	1.024334×10^{0}	1.811×10^{-4}	1.768×10^{-4}	4
(0,1,1,2)	3.755280×10^{-1}	3.758095×10^{-1}	2.815×10^{-4}	7.491×10^{-4}	4
(1,0,1,2)	3.759796×10^{-1}	3.758095×10^{-1}	1.701×10^{-4}	4.526×10^{-4}	4
(1,1,1,2)	1.142660×10^{-1}	1.142569×10^{-1}	9.023×10^{-6}	7.897×10^{-5}	6
(0,0,2,3)	8.245848×10^{-1}	8.247579×10^{-1}	1.731×10^{-4}	2.098×10^{-4}	4
(0,1,2,3)	3.081561×10^{-1}	3.074143×10^{-1}	7.418×10^{-4}	2.413×10^{-3}	4
(1,0,2,3)	3.077714×10^{-1}	3.074143×10^{-1}	3.571×10^{-4}	1.162×10^{-3}	4
(1,1,2,3)	9.653955×10^{-2}	9.614571×10^{-2}	3.938×10^{-4}	4.096×10^{-3}	4

All integrals reproduced to **4–6 significant digits**.
Maximum relative error $\approx 4\times10^{-3}$.