



AI-Based C++ Code Generation for Fitting Models in FITTER_WEB

A. G. Soloviev, T. M. Solovjeva, K. V. Lukyanov

What is fitting?

- ❑ Process of optimizing model parameters to describe experimental data (spectra, particle distributions).
- ❑ Quality measured by χ^2 (chi-squared):

$$\chi^2 = \frac{1}{N - N_{par}} \sum_{i=1}^N \left(\frac{f(x_i) - y_i}{\Delta y_i} \right)^2$$

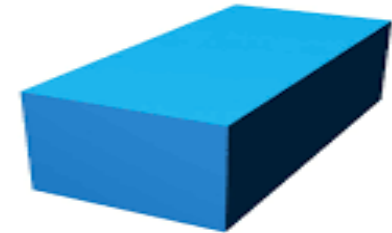
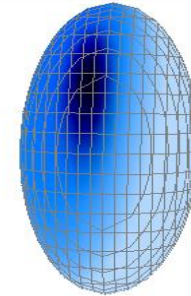
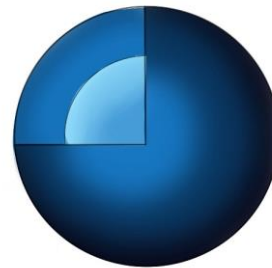
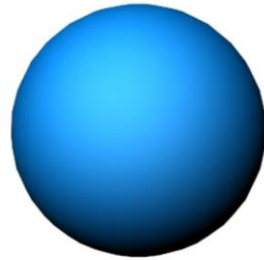
- N - Number of data points
- N_{par} - Number of model parameters
- $f(x_i)$ - Model prediction
- $y_i \pm \Delta y_i$ - Measured values with errors

How physicists use fitting?

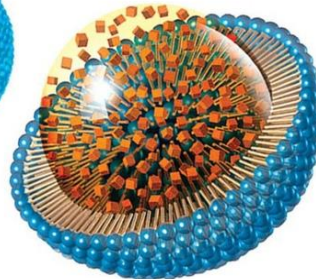
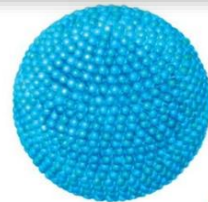
- ❑ Test hypotheses by comparing models with experimental data.
- ❑ Extract physical parameters (e.g., radii, amplitudes) from noisy data.
- ❑ Minimize χ^2 to ensure scientific accuracy.

Small-Angle Scattering: Key Model Types

Classical Form Factors integrated both in FITTER and FITTER_WEB



Vesicles & Membranes integrated in FITTER_WEB



Soloviev A.G., Solovjeva T.M.,
Lukyanov K.V., Zemlyanaya E.V.
SITITO (2024) V. 20, N. 3,

MORE MODELS

Рентгеновское и
нейтронное малоугловое
рассеяние / Дмитрий
Иванович Свергун, Лев
Абрамович Фейгин. – М.
: Наука, 1986. – 279 с



History of FITTER application

- **2003-2008**

Fitter is a standalone C++ program on Linux\Windows aimed to fit a chosen theoretical multi-parameter function through a set of data points. Respective theoretical models are implemented in it.

- **2012** Additional theoretical models implemented.

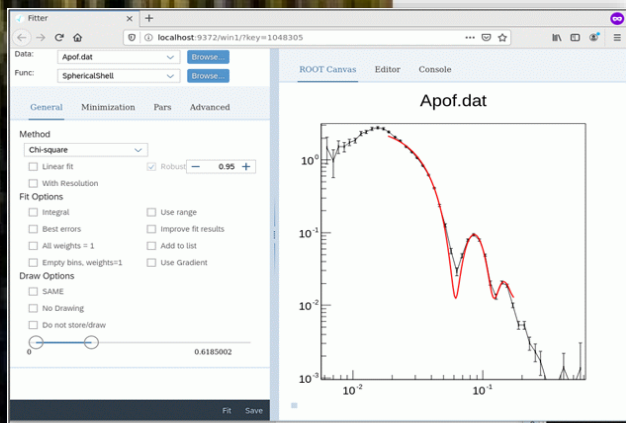
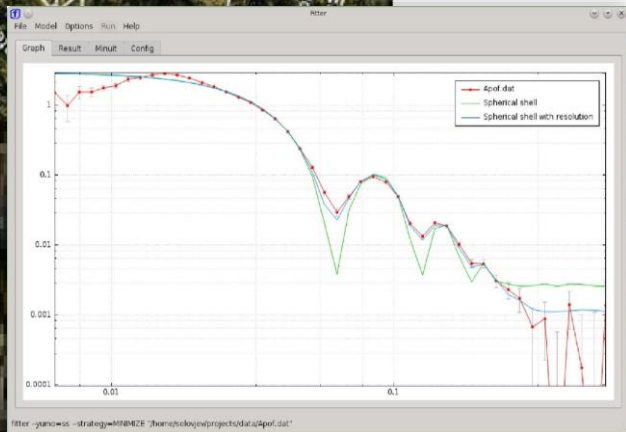
- **2022**

Web-version of FITTER was developed as a standalone web-service.

- **2023-2024**

Deploying and optimizing FITTER_WEB in JINR cloud infrastructure.

- **2025 – WHO WILL CODE THE MODELS?..**



How AI Simplifies Model Implementation

CORE IDEA:

Physicists describe models in LaTeX $\Rightarrow$ AI generates optimized C++ code

User Input:

- ☐ LaTeX formula (e.g., scattering model)
- ☐ Parameter constraints

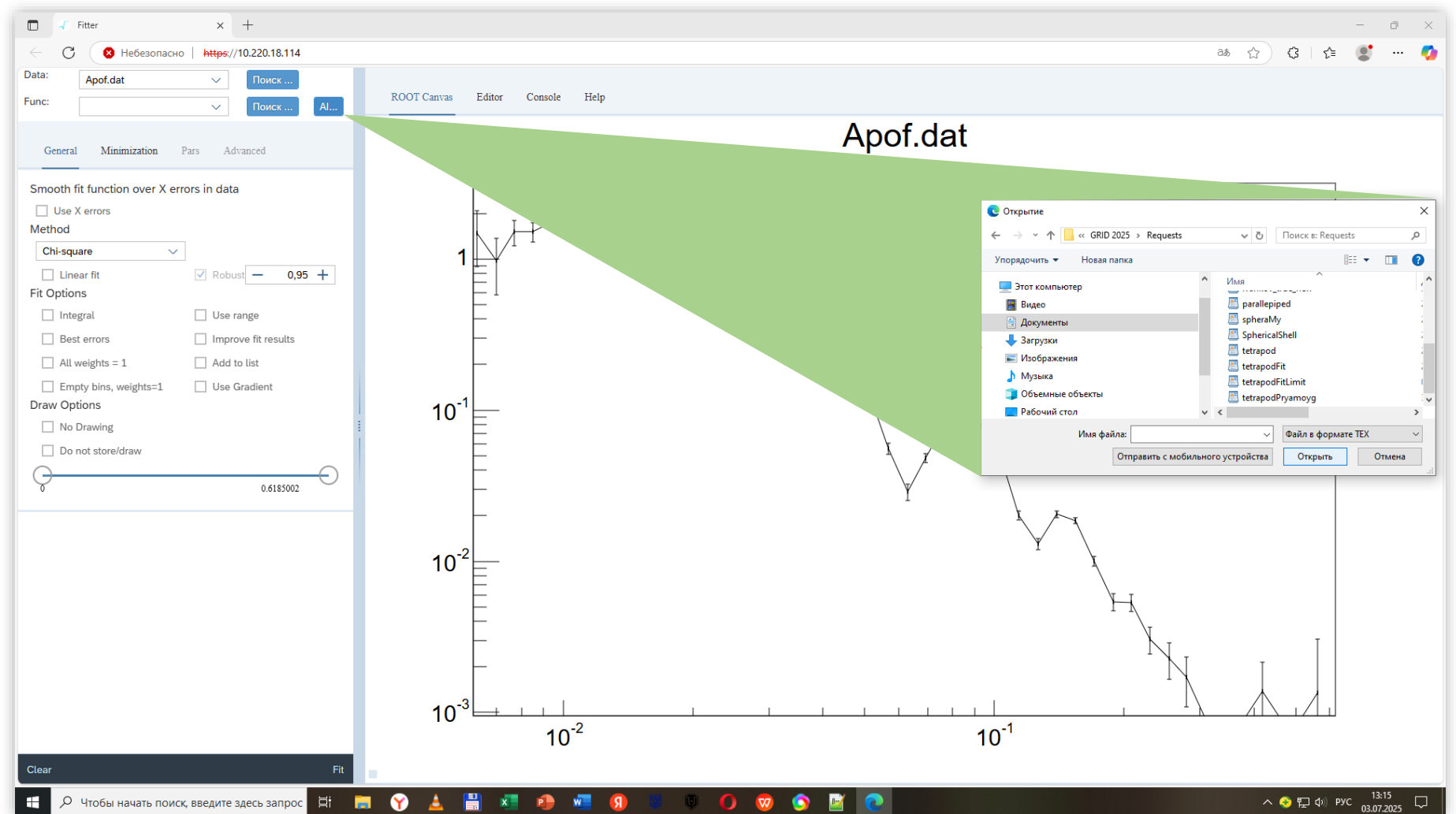
AI Output:

- ☐ Ready-to-use TF1 ROOT function
- ☐ or editable prototype

Current prompt (subject to further studying)

Convert the following LaTeX formulas to C++ code to create a TF1 function in ROOT:
The original LaTeX is unchanged as a comment
The function is strictly in the format 'TF1* getFCN()'
All auxiliary calculations are inside getFCN() as lambdas
For a one-dimensional integral, use ROOT::Math::Integrator
For a multidimensional integral, use ROOT::Math::IntegratorMultiDim
Detailed parameter comments
Here is the conversion formula: **latex code of formula with parameters here**

New FITTER_WEB workflow





A LaTeX file with example formula

$$I(Q) = A \left[\Phi(QR_1) - \left(\frac{R_2}{R_1} \right)^3 \Phi(QR_2) \right]^2 + B$$

$$\Phi(t) = 3 \frac{\sin t - t \cos t}{t^3}$$

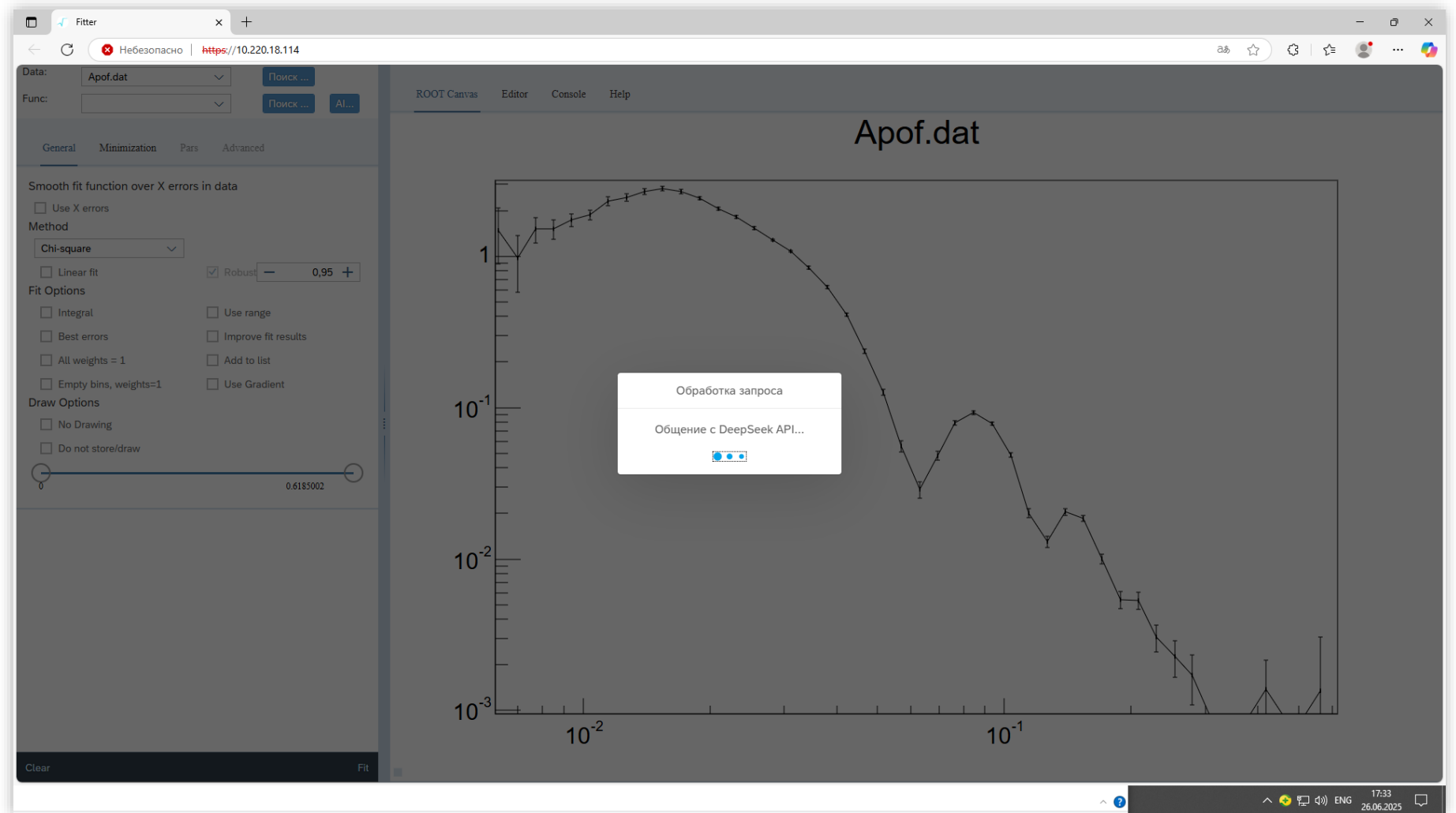
`{\bf Spherical shell}` of the outer radius R_1 and the inner radius R_2

```
\begin{eqnarray*}
I(Q) &=& A \left[ \Phi(QR_1) - \right. \\
&& \left. \left( \frac{R_2}{R_1} \right)^3 \Phi(QR_2) \right]^2 + B \\
\Phi(t) &=& 3 \frac{\sin t - t \cos t}{t^3} \\
\end{eqnarray*}
```

Initial values for the parameters are: $R_1 = 65$, $R_2 = 20$
 The parameter values R_1 , R_2 and A are strictly greater than zero.



AI request processing



The code appears in the Editor tab

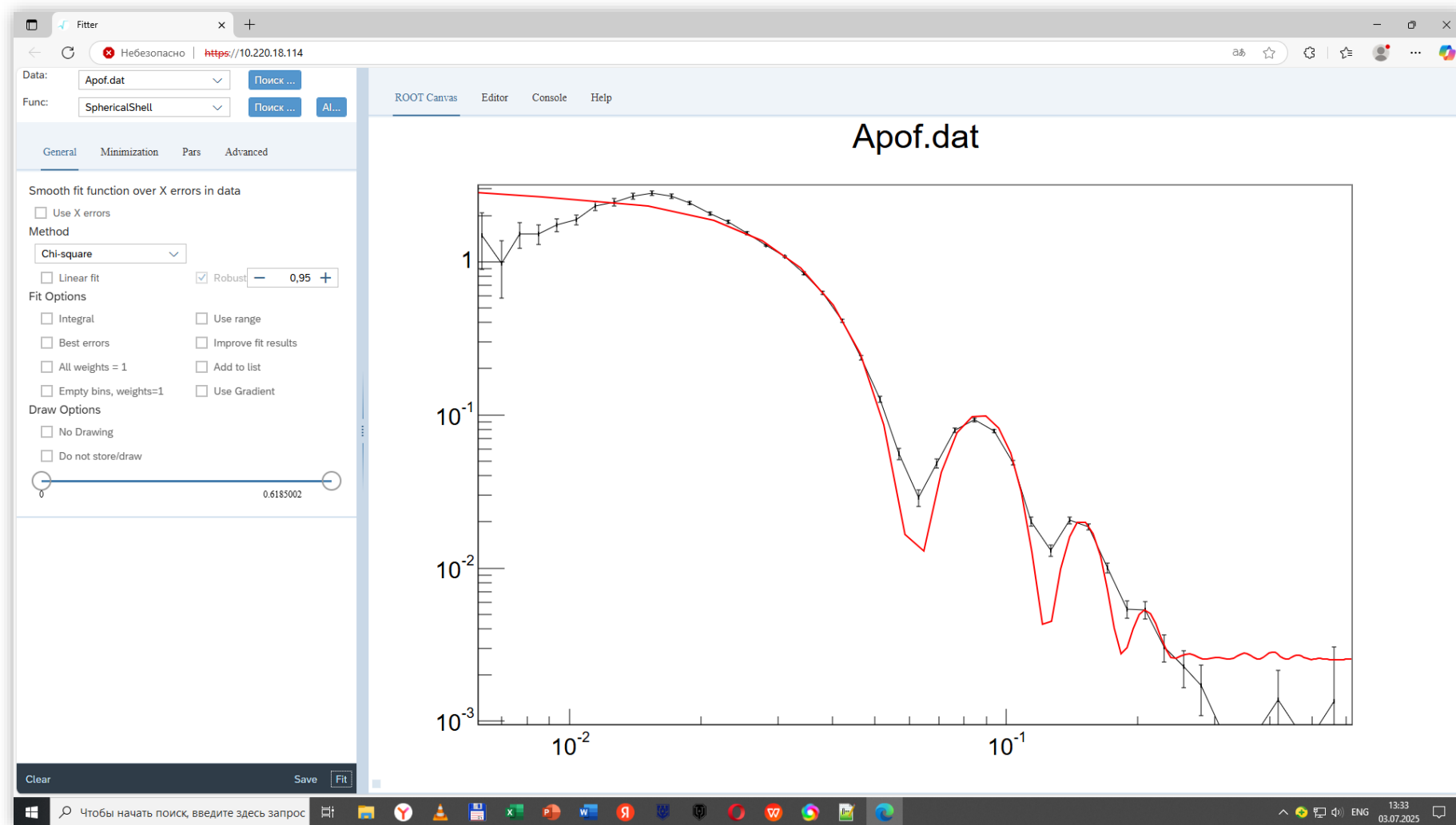
The screenshot displays the Fitter application window. The left sidebar contains settings for data (Apof.dat), function (SphericalShell), and various fit options. The main area is divided into a ROOT Canvas and an Editor tab. The Editor tab shows the following C++ code:

```

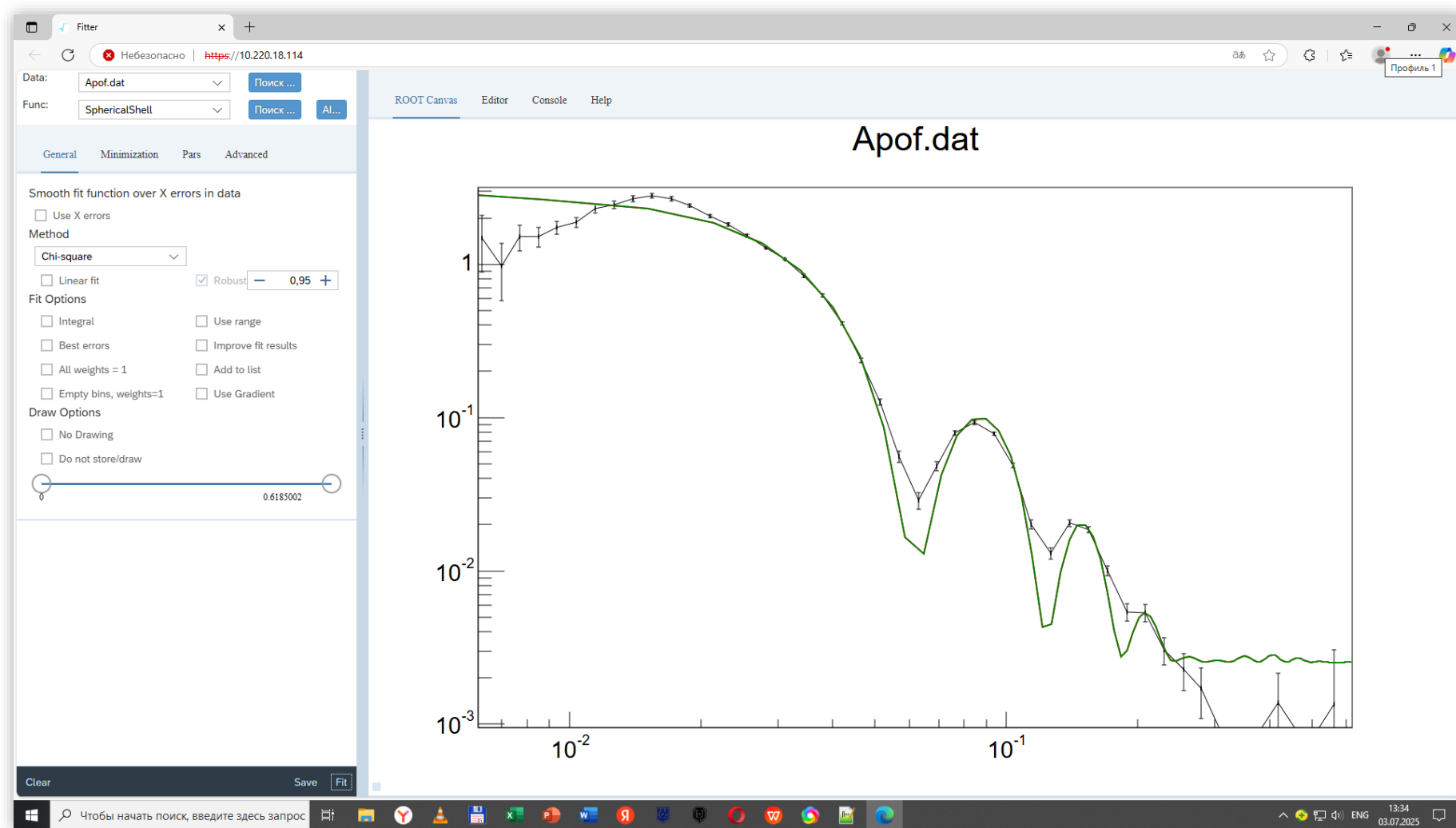
1 // Формула:
2 // Spherical shell of the outer radius $R_1$ and the inner radius $R_2$
3 // \begin{eqnarray*}
4 // I(Q) \hat{=} \frac{1}{Q} \left[ \frac{1}{Q} \left( \frac{R_2}{R_1} \right)^3 \frac{1}{Q} \left( \frac{R_2}{R_1} \right)^3 \frac{1}{Q} \left( \frac{R_2}{R_1} \right)^3 \right]^2 + B \\
5 // A \cdot \frac{1}{Q} \left( \frac{R_2}{R_1} \right)^3 \frac{1}{Q} \left( \frac{R_2}{R_1} \right)^3 \frac{1}{Q} \left( \frac{R_2}{R_1} \right)^3 \\
6 // \end{eqnarray*}
7 // Initial values: $R_1 = 65$, $R_2 = 20$
8
9
10 TF1* getFCN() {
11 // Вспомогательная функция Phi(t) = 3*(sin(t) - t*cos(t))/t^3
12 auto phi = [](double t) -> double {
13     if (std::abs(t) < 1e-10) return 1.0; // Предел при t->0
14     return 3.0 * (std::sin(t) - t*std::cos(t)) / (t*t*t);
15 };
16
17 // Основная функция I(Q) = A * [Phi(Q*R1) - (R2/R1)^3 * Phi(Q*R2)]^2 + B
18 auto mainFunc = [phi](double* x, double* par) -> double {
19     const double Q = x[0]; // Входная переменная (волновой вектор)
20
21     // Параметры:
22     const double A = par[0]; // Амплитуда
23     const double B = par[1]; // Фон
24     const double R1 = par[2]; // Внешний радиус
25     const double R2 = par[3]; // Внутренний радиус
26
27     const double t1 = Q * R1;
28     const double t2 = Q * R2;
29     const double ratio = R2 / R1;
30
31     const double term = phi(t1) - std::pow(ratio, 3) * phi(t2);
32     return A * term * term + B;
33 };
34
35 // Создание TF1 функции с 4 параметрами
36 TF1* func = new TF1("SphericalShell", mainFunc, 0, 200, 4);
37
38 // Установка начальных значений параметров
39 func->SetParameter(0, 1.0); // A: начальное значение амплитуды
40 func->SetParameter(1, 0.0); // B: начальное значение фона
41 func->SetParameter(2, 65.0); // R1: внешний радиус = 65
42 func->SetParameter(3, 20.0); // R2: внутренний радиус = 20
43
44 // Именование параметров для удобства
45 func->SetParName(0, "A");
46 func->SetParName(1, "B");
47 func->SetParName(2, "R1");
48 func->SetParName(3, "R2");
49
50 return func;
51 }

```

Fitting by function from AI



Fitting a function written by the user



Fitting Results

Fitting a function generated by AI

```
*****
Minimizer is Minuit / Migrad
Chi2          = 688.425
NDf           = 42
Edm           = 9.55216e-09
NCalls        = 215
R1            = 61.1049 +/- 0.190481
R2            = 39.2687 +/- 0.256541
A             = 5.34815 +/- 0.0950282
B             = 0.00253013 +/- 0.000207522
```

Fitting a function written by the user

```
*****
Minimizer is Minuit / Migrad
Chi2          = 688.425
NDf           = 42
Edm           = 9.5182e-09
NCalls        = 215
Outer radius  = 61.1049 +/- 0.190485
Inner radius  = 39.2687 +/- 0.256545
Amplitude     = 5.34815 +/- 0.0950293
Background    = 0.00253013 +/- 0.000207522
```

Code comparison

user

```
TF1* getFCN() {
    auto phi = [](double t) -> double {
        if (std::abs(t) < 1e-12) return 1.0; // Предел при t→0
        double sin_t = std::sin(t);
        double cos_t = std::cos(t);
        return 3.0 * (sin_t - t * cos_t) / (t * t * t);
    };
    // Основная функция I(Q) = A [Φ(QR1) - (R2/R1)3 Φ(QR2)]2 + B
    auto mainFunc = [phi](double* x, double* p) -> double {
        // Параметры:
        // p[0] = A (амплитуда)
        // p[1] = B (фон)
        // p[2] = R1 (внешний радиус)
        // p[3] = R2 (внутренний радиус)
        double Q = x[0]; // Переменная Q
        double QR1 = Q * p[2];
        double QR2 = Q * p[3];
        double ratio = p[3] / p[2];
        double ratio3 = ratio * ratio * ratio;
        double term = phi(QR1) - ratio3 * phi(QR2);
        return p[0] * term * term + p[1];
    };
    // Создание TF1 функции с 4 параметрами
    TF1* func = new TF1("SphericalShell", mainFunc, 0, 100, 4);
    // Установка начальных значений параметров
    func->SetParameter(0, 1.0); // A = 1.0 (временное значение)
    func->SetParameter(1, 0.0); // B = 0.0 (временное значение)
    func->SetParameter(2, 65.0); // R1 = 65
    func->SetParameter(3, 20.0); // R2 = 20
    // Именованые параметры для удобства
    func->SetParName(0, "A");
    func->SetParName(1, "B");
    func->SetParName(2, "R1");
    func->SetParName(3, "R2");
    return func;
}
```

```
#include <TF1.h>
TF1* getFCN(){
    auto fcn = [] (Double_t * x, Double_t * p)
    {
        auto phi = [] ( const Double_t & x )
        {
            Double_t res = 1;
            if ( x )          res = 3 * ( sin(x) - x * cos(x) ) / ( x * x * x );
            return res;
        };
        Double_t q = x[0];
        Double_t r2r1 = p[1]/p[0];
        Double_t res = phi ( q * p[0] ) - phi ( q * p[1] ) * ( r2r1 * r2r1 * r2r1 );
        return p[2] * res * res + p[3];
    };
    TF1* modelFCN = new TF1("SphericalShell", fcn, 0, 1, 4);
    modelFCN->SetParNames("Outer radius", "Inner radius", "Amplitude",
    "Background");
    modelFCN->SetParameters(65, 20, 1, 0);
    return modelFCN;}

```



AI



Possible benefits of AI in our app

- ☐ Reduced time and resource costs.
- ☐ Physicists and biologists can focus on the challenges of scientific problems without spending their time learning C++.
- ☐ Artificial intelligence analyze large amounts of data, so it offers optimal approaches for coding. Its work is based on trained models that recognize patterns in millions of lines of code.
- ☐ Fewer errors: typos and syntax issues.
- ☐ Quick prototyping.



Current problems to be solve

- ☐ Switch to the home neural network deployed on the LIT servers.
- ☐ Select the optimal LLM (large language model).
- ☐ Optimize the prompt to ensure the repeatability of the generated model code.
- ☐ Extend the fitter so that it can accept files with the extension .pdf.
- ☐ Develop an interface so that the user can add additional instructions to the prompt located inside the fitter.



Thanks for your attention