Applicability of the Julia programming language for mass solving of systems of differential equations using the example of the problem of modeling magnetic moment reversal in the φ_0-Josephson junction

Maxim V. Bashashin[1,2,*], Ilhom R. Rahmonov[1,2,3] and Elena V. Zemlyanaya[1,2]

[1] *Joint Institute for Nuclear Research, 6 Joliot-Curie St, Dubna, 141980, Russian Federation*

[2] *Dubna State University, 19 Universitetskaya St, Dubna, 141980, Russian Federation*

[3] *Moscow Institute of Physics and Technology Nauchny pereulok 9, Dolgoprudny, 141701, Russian Federation*

Abstract

The possibilities of the Julia programming language for numerical study of multiparameter models described by systems of nonlinear differential equations are considered. As an example, we use the model of a point φ_0-Josephson junction of the "superconductor - ferromagnet - superconductor" type with direct connection between the magnetic moment and the Josephson current. The methodology and features of numerical solution of systems of equations based on the Julia language are tested. The possibilities of the Julia language for implementing parallel computations when calculating the regions of magnetic moment reversal in a wide range of parameters are presented.

Keywords

Julia, Python, Josephson junction, mathematical modelling, parallel computing

1. Introduction

The Julia programming language [1] is declared as a tool oriented towards mathematical calculations, which is of interest to the researchers for the purpose of using it for their studies. Of greatest interest are the possibilities of solving systems of differential equations, as well as the instruments for high performance computing.

To test the capabilities of the Julia language, we consider the previously solved problem on indication of the magnetic moment reversal domains in φ_0-Josephson junctions with spin-orbit coupling in a ferromagnetic layer.

2. Mathematical model

Schematic view of the φ_0-junction under consideration is shown in Figure 1. The easy axis of the ferromagnetic layer is directed along the z-axis, which also coincides with the direction of the spin-orbit potential gradient. The component of the magnetic moment m_y is associated with the superconducting (Josephson) current, which is directed along the x-axis.

The dynamics of the magnetic moment of such junction is described by the following equation [2]:

$$\frac{d\vec{m}}{dt} = -\frac{\omega_F}{1+\vec{m}^2\alpha^2}\{[\vec{m}\times\vec{H}] + \alpha[\vec{m}(\vec{m}\vec{H}) - \vec{H}\vec{m}^2]\} \qquad (1)$$

Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems 2025 (ITTMM 2025), Moscow, April 07–11, 2025

*Corresponding author.

bashashinmv@jinr.ru (M. V. Bashashin); rahmonov@theor.jinr.ru (I. R. Rahmonov); elena@jinr.ru (E. V. Zemlyanaya)

0000-0002-2706-8668 (M. V. Bashashin); 0000-0001-8149-9533 (E. V. Zemlyanaya)

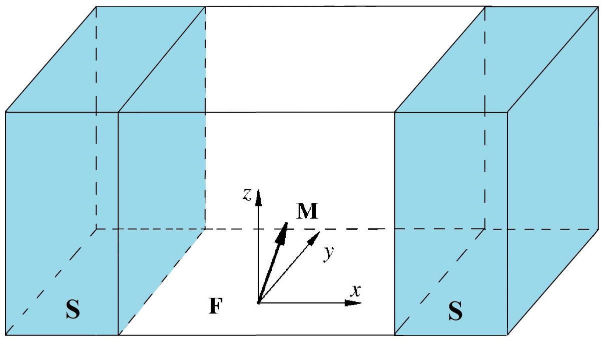


Figure 1: Schematic view of the φ_0-junction [2]; **S** and **F** are the superconducting and ferromagnetic layers, respectively, **M** is the magnetic moment vector of the ferromagnetic layer, the easy axis of which is directed along the z-axis. The external current pulse is directed along the x-axis.

where α is the damping parameter, ω_F is the normalized ferromagnetic resonance frequency, $\vec{H}$ is the effective magnetic field with components:

$$\begin{aligned} H_x(t) &= 0; \\ H_y(t) &= Gr\sin(\varphi(t) - rm_y(t)); \\ H_z(t) &= m_z(t). \end{aligned} \tag{2}$$

Here G is the ratio of the Josephson energy to the magnetic anisotropy energy, r is the spin-orbit coupling parameter, $m_{x,y,z}$ are the x, y, z components of the magnetic moment $\vec{m}$. All quantities are reduced to the dimensionless form as shown in [2].

The Josephson phase difference φ can be found using this equation:

$$\frac{d\varphi}{dt} = \frac{1}{\omega}(I_{pulse}(t) - \sin(\varphi - rm_y)), \tag{3}$$

where ω– Josephson frequency. A rectangular pulse with amplitude A_s and duration Δt was used as I_{pulse}:

$$I_{pulse}(t) = \begin{cases} A_s, & t \in [t_0 - 1/2\Delta t, t_0 + 1/2\Delta t]; \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The initial conditions are as follows:

$$m_x(0) = 0, \quad m_y(0) = 0, \quad m_z(0) = 1, \quad \varphi(0) = 0. \tag{5}$$

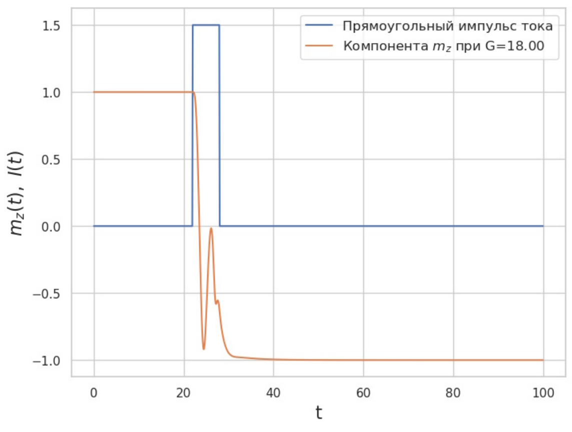
3. Features of Julia-based implementation

To solve ODE systems using the Julia language, the DifferentialEquations.jl [3] module should be used, which includes more than two hundred methods (solvers).

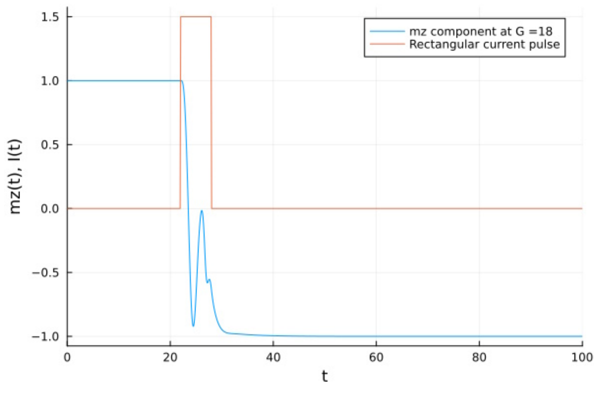
The function has to be specified that describes the right-hand sides of all equations of the ODE system under study. Then, the initial condition vectors, the integration domain vector, and the parameter vector are formed. Based on this data, a special object called ODEProblem is formed. Then the object is passed to the solver with desirable parameters of absolute and relative tolerance in the adaptive timestepping. It is important to note that the numerical method for solving the system is selected automatically.

To check the correctness of the solution obtained by means of the Julia code, these results were compared with the results obtained within the Python implementation of this task in [4]. For this purpose, the function $m_z(t)$ was launched with two parameters of G: G=18 and G=8.

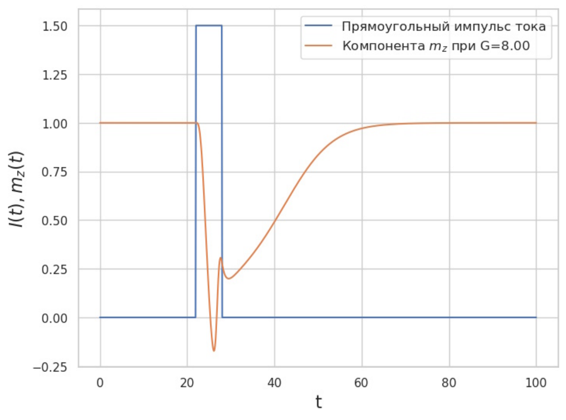
Other parameters of the model are following: r=0.1, α=0.1, A_s=1.5, Δ_t=6, t_0=0, t_{max}=100. The results are presented in Figure 2. Panels (a,c) correspond the Python calculations while panels (b,d) show the Julia results. The case G=18 corresponds the effect of magnetic moment reversal when function $m_z(t)$ changes in simulations from +1 to −1. In case G=8 the magnetic moment reversal does not occur. One can see, the Python- and Julia-simulation results match for both calculations, which confirms the correctness of the solution based on the Julia language.



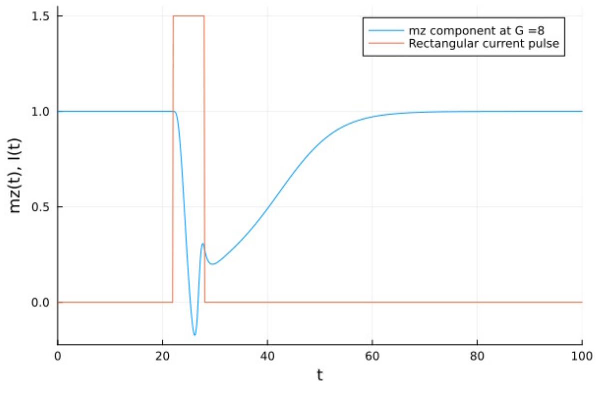
(a)



(b)



(c)



(d)

Figure 2: Comparison of simulation results using Python (a),(c) and Julia (b),(d) programming languages. Calculation parameters: G=18 for (a),(b) and G=8 for (c),(d), r=0.1, α=0.1, A_s=1.5, Δ_t=6, t_0=0, t_{max}=100.

4. Parallelism in multiple calculations

To find magnetic moment reversal domains, the initial value problem (1–5) must be solved many times in a wide range of parameters of the model. For this purpose, the DifferentialEquations.jl module provides a mechanism for multiple solution of the ODE system. The existing function with the right-hand sides of the ODEs is supplemented with the functions of the “modifier of the problem conditions” and the “output modifier function”. For each individual solution, its own set of parameters is formed and the modifier function will substitute this set for the solution of ODE system. The output modifier function allows one to introduce conditions for saving data based on the results of solving a group of problems. For example, in case we are only interested in the final value of the m_z component, the output modifier will save only this value. Based on the functions described above, an EnsembleProblem object is formed and solved in a way similar to single ODE systems, but with the addition of a parameter responsible for the number of iterations.

Table 1
Times (in seconds) of multi-thread execution of Python and Julia codes in case of numerical solving the system (1–5) for 10000 sets of parameters.

Number of threads	Python+Jupiter	Julia AMD	Julia Intel
1	2374.26	19.35	27.63
2	1221.51	9.1	19.21
4	622.15	5.76	13.15
6	429.32	4.87	13.3

Multiple solutions of the ODE systems lead to significant time costs when calculating in the serial mode. The Julia tool provides a parallel calculation for this this type of tasks.

To arrange parallel calculations, the multi-threading is used through a mechanism called EnsembleThreads. This mechanism automatically distributes single tasks between a specified number of threads. Using this approach, a comparative test of parallel Python and Julia implementations was carried out when calculating the ODE system for 10000 sets of parameters. The parallel Python implementation was performed in the Jupyter Notebook environment on Intel Xeon CPU E5-2680 v3 processor, and the parallel Julia implementation was performed in the Windows environment on AMD Ryzen 7 7800X3D and Intel Core i7 8550U processors.

The test results presented in Table 1 show the superiority of the Julia language over the Python language in solving the presented problem. The high performance of the Julia language is ensured by pre-compiled code execution and widely implemented automation in the calculation process, which allows for the optimal selection of methods for executing user code.

5. Conclusions

Using the representative problem of determining magnetic moment reversal domains within the φ_0-Josephson junction model, the applicability of the Julia language was tested for the problems requiring multiple solutions of ODE systems. For this problem, previously obtained numerical results were reproduced, that confirmes correctness of the Julia implementation. Comparative analysis of Python and Julia implementations showed an advantage of the Julia language which provides a 100 times speedup of parallel computing. Note also the simplicity of parallel implementation, ability to use graphic accelerator in calculations and the possibility to integrate the Julia applications into interactive environments, such as Jupyter Notebook. This opens perspectives of further use of Julia tool in numerical studies of complex multi-parameter problems requiring solutions of systems of differential equations.

Author Contributions: Computer implementations, calculations, writing the paper — M.V. Bashashin; formulae, editing the text — I.R. Rahmonov; analysis of results, writing and editing the paper — E.V. Zemlyanaya. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Russian Science Foundation grant № 22-71-10022.

Data Availability Statement: Data sharing is not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Acknowledgments: We thank the HybriLIT team for the access to the Jupyter Notebook system.

References

1. Bezanson, J., Edelman, A., Karpinski, S. & Shah, V. B. A Fresh Approach to Numerical Computing. *SIAM Review* **59,** 65–98. doi:10.1137/141000671 (**january** 2017).

2. Atanasova, P. K., Panayotova, S. A., Rahmonov, I. R., Shukrinov, Y. M., Zemlyanaya, E. V. & Bashashin, M. V. Periodicity in the Appearance of Intervals of the Reversal of the Magnetic Moment of a φ_0 Josephson Junction. *JETP Letters* **110,** 722–726. doi:10.1134/S0021364019230073 (**november** 2019).

3. Rackauckas, C. & Nie, Q. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software* **5,** 1–10. doi:10.5334/jors.151 (**january** 2017).

4. Bashashin, M. V., Butenko, Y. A., Kulikov, K. K., Nechaevskiy, A. V., Rahmonov, I. R., Rahmonova, A. R., Streltsova, O. I. & Zuev, M. I. *Инструментарий для моделирования гибридных наноструктур сверхпроводник/магнетик* (JINR, http://studhub.jinr.ru:8080/books/intro.html, 2022).