



## Осенняя Школа по информационным технологиям ОИЯИ 2024



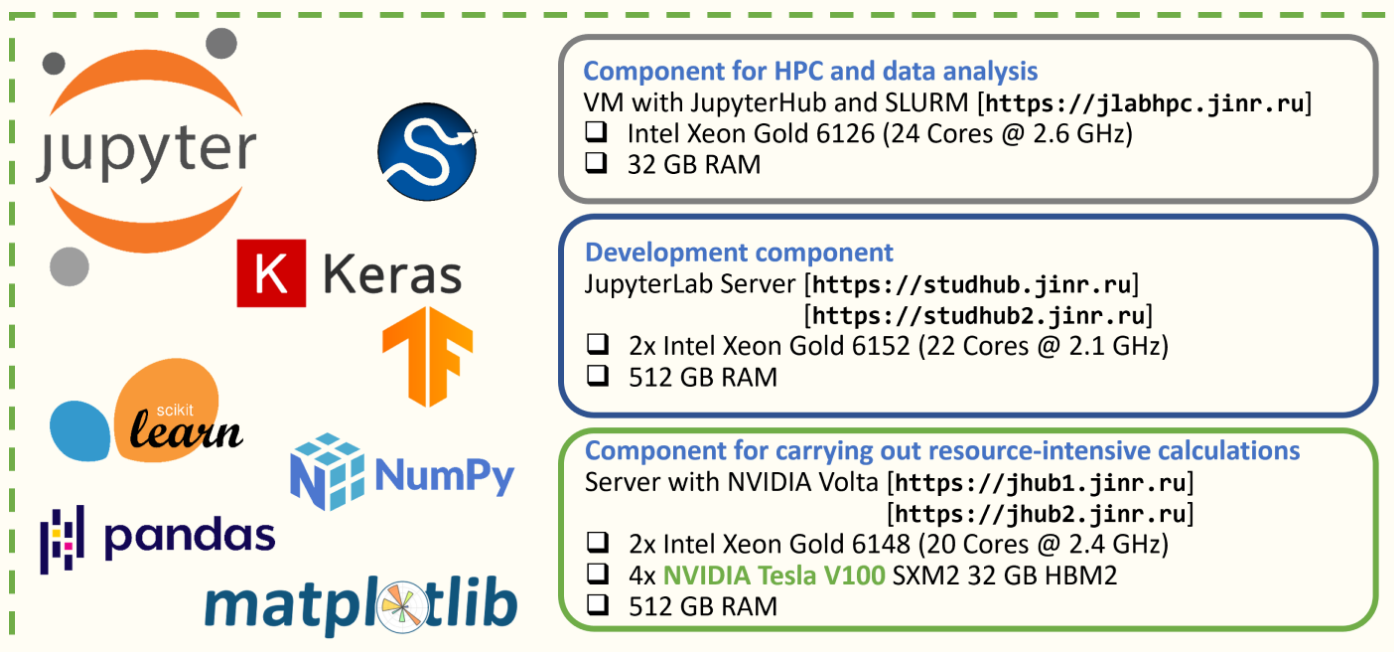
Практическое занятие

# Инструментарий на основе Python-библиотек и экосистемы Jupyter для задач математического моделирования систем, основанных на джоозефсоновских переходах



Аникина А.И., Башакин М.В., Бежанян Т.Ж., Беляков Д.В., Воронцов А.С., Зуев М.И., Кокаев Д.А., Кокорев А.А., Нечаевский А.В., Пряхина Д.И., Рахмонов И.Р., Рахмонова А.Р., Стрельцова О.И., Шадмехри С.

Экосистема ML/DL/HPC гетерогенной платформы HybriLIT [<http://hlit.jinr.ru>]

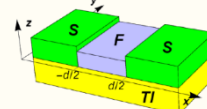
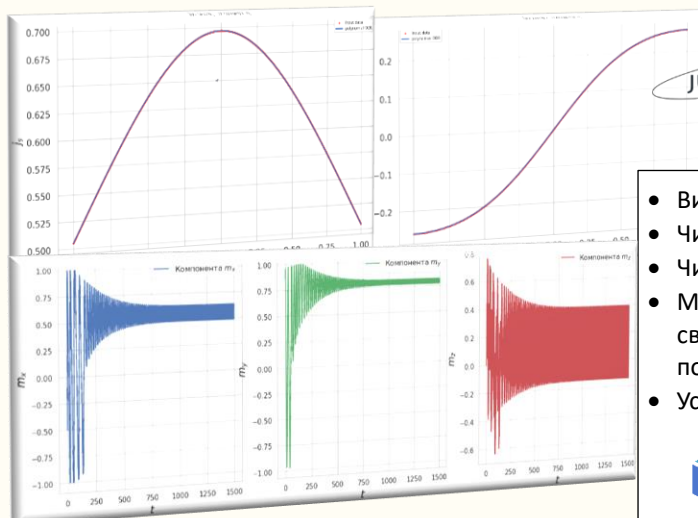


The diagram illustrates the HybriLIT ecosystem components, categorized into three main functional areas:

- Component for HPC and data analysis:**
  - VM with JupyterHub and SLURM [<https://j1labhpc.jinr.ru>]
  - Intel Xeon Gold 6126 (24 Cores @ 2.6 GHz)
  - 32 GB RAM
- Development component:**
  - JupyterLab Server [<https://studhub.jinr.ru>], [<https://studhub2.jinr.ru>]
  - 2x Intel Xeon Gold 6152 (22 Cores @ 2.1 GHz)
  - 512 GB RAM
- Component for carrying out resource-intensive calculations:**
  - Server with NVIDIA Volta [<https://jhub1.jinr.ru>], [<https://jhub2.jinr.ru>]
  - 2x Intel Xeon Gold 6148 (20 Cores @ 2.4 GHz)
  - 4x NVIDIA Tesla V100 SXM2 32 GB HBM2
  - 512 GB RAM

Software libraries and frameworks shown include Jupyter, Keras, scikit-learn, NumPy, pandas, and matplotlib.

### План занятия



- Визуализация в Python – библиотеки **matplotlib**, **seaborn**
  - Численное интегрирование и аппроксимация
  - Численное решение задачи Коши – библиотека **SciPy**
  - Математическое моделирование джоозефсоновского перехода сверхпроводник/ферромагнетик/сверхпроводник на поверхности трехмерного топологического изолятора
  - Ускорение вычислений – библиотека **Joblib**
- 

# Исследование поведения подинтегральной функции при различных значениях параметров, численное интегрирование и аппроксимация интегралов при различных значениях параметров

Рассмотрим вычисление интеграла:

$$j_s = \int_{-\pi/2}^{\pi/2} \cos \phi \exp\left(-\frac{\bar{d}}{\cos(\phi)}\right) \cos(rm_x \tan \phi) d\phi$$

Подынтегральная функция на концах интервала интегрирования не определена, однако

$$\lim_{\phi \rightarrow \pm \frac{\pi}{2}} f(m_x, r, d, \phi) = 0.$$

Параметры модели:

- $\bar{d}$  – безразмерная длина контакта,  $\bar{d} \in [0.1, 0.8]$ ;
- $r$  – безразмерный параметр, определяющий величину спин-орбитального взаимодействия,  $r \in [0.1, 2]$ ;
- $G$  – отношение энергии Джозефсона к энергии магнитной анизотропии,  $G \in [0.1, 10]$ ;
- $\alpha$  – диссипация Гилберта,  $\alpha \in [0.01, 0.2]$ ;
- $\omega_F = 1$ .

Функция, подлежащая определению, значения которой при вычислении интеграла играет роль параметра

- $m_x$  – компонента намагниченности.

• **Создание функции:** определение подинтегральной функции

```
def funct_js(phi, mx, r, d):
    ''' Defines the integrand in the
    definition of current js, mx, r, d -
    parameters '''
    return (np.cos(phi) *
            np.exp(-d / np.cos(phi)) *
            np.cos(r*mx*np.tan(phi)))
```

• **Расчет** при следующих значениях параметров:

```
d = 0.8
r = 1.1
mx = 0.5
```

• **Определение массива:**

```
phi = np.linspace(-np.pi/2, np.pi/2, 300,
                  endpoint=True)
```

**Построение графиков:** библиотека matplotlib

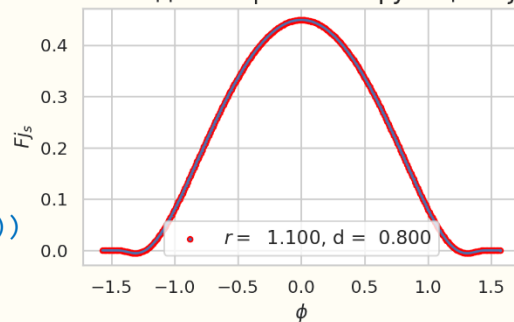
```
# подключение библиотеки matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

y = funct_js(phi, mx, r, d)

# размеры графика
plt.figure(figsize=(5, 3), dpi=200)
# отрисовка графика по координатам x и y
plt.scatter(phi, y, edgecolor="red", s=10,
            label= r'$r = $ %6.3f, d = %6.3f'%(r,d))
plt.plot(phi, y)
plt.xlabel("$\phi$")
plt.ylabel("$j_{s}$")
plt.title("Зависимость $j_{s}$ от угла $\phi$")
plt.legend()

plt.show()
```

Зависимость подинтегральной функции  $F_{j_s}$  от угла  $\phi$



**Численное интегрирование с использованием библиотеки SciPy**

```
# подключение библиотеки scipy
from scipy.integrate import quad
```

```
%%time
js = quad(func_tjs, -np.pi/2, np.pi/2, args=(mx,r,d))
```

Воспользуемся функцией `quad` для вычисления определенного интеграла:  
`quad(func, a, b, args=(), full_output=0, epsabs=1.49e-08, epsrel=1.49e-08, limit=50, points=None, weight=None, wvar=None, wopts=None, maxp1=50, limlst=50, complex_func=False)`  
 Функция для интегрирования от  $a$  до  $b$  (возможно, с бесконечными пределами) на основе библиотеки Фортрана QUADPACK.

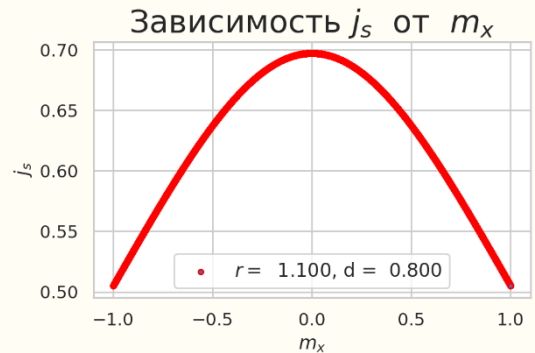
**Время интегрирования (все параметры зафиксированы, кроме  $m_x$ )**

Библиотека NumPy добавляет поддержку больших многомерных массивов и матриц, вместе с большим набором высокоуровневых математических функций.

```
# подключение библиотеки
import numpy as np

Npoint = 1000 # количество точек
# создание последовательности данных, равномерно
# расположенных на числовой прямой в интервале (-1, 1)
arr_mx = np.linspace(-1, 1, Npoint, endpoint=True)
# создания нулевых матриц
arr_js = np.zeros(Npoint, dtype=np.float64)
arr_err = np.zeros(Npoint, dtype=np.float64)
```

```
%%time
for ind in range(Npoint):
    mx = arr_mx[ind]
    arr_js[ind], arr_err[ind] = quad(func_tjs, -np.pi/2, np.pi/2, args=(mx,r,d))
```



**Интерактивное управление в Jupyter Notebook – библиотека ipywidgets**

```
# подключение библиотеки ipywidgets
import ipywidgets as widgets
from ipywidgets import interact, interact_manual, Label
%matplotlib widget
@interact
def show_js_mx(mx=(-1.0, 1.0, 0.1), r=(0.1, 2.0, 0.1),
              d=(0.1, 0.8, 0.1)):
    phi = np.linspace(-np.pi/2, np.pi/2, 300,
                      endpoint=True)
    fig = plt.figure(figsize=(8, 6))
    plt.scatter(phi, func_tjs(phi, mx, r, d),
                edgecolor="red", s=10,
                label=r'$r = $ %6.3f,
                    d = %6.3f'%(r,d))

    plt.plot(phi, func_tjs(phi, mx, r, d))
    plt.xlabel("$\phi$")
    plt.ylabel("$j_{s}$")
    plt.title("Зависимость $f$ от угла $\phi$")
    plt.legend(loc='upper right')

plt.show()
```



Для дальнейших исследований представляет интерес рассмотрение поведения функции тока  $j_s$  и интегралов  $I_x$  и  $I_y$  при различных значениях параметров.

## Поведение функции тока $j_s$ и интегралов $I_x, I_y$ при различных значениях параметров

Поведение функции тока  $j_s$  при различных значениях параметров:

```
@interact
def show_func_js(r=(0.1, 2.0, 0.1), d=(0.1, 0.8, 0.1)):
    Npoint = 1000 # количество точек (число вызовов функции интегрирования)
    arr_mx = np.linspace(-1, 1, Npoint, endpoint=True)
    arr_js = np.zeros(Npoint, dtype = np.float64)
    arr_err = np.zeros(Npoint, dtype = np.float64)

    # вычисление интегралов
    for ind in range(Npoint):
        mx = arr_mx[ind]
        # интегрируем для каждого
        arr_js[ind], arr_err[ind] = quad(
            funct_js, -np.pi/2,
            np.pi/2,
            args=(mx,r,d))

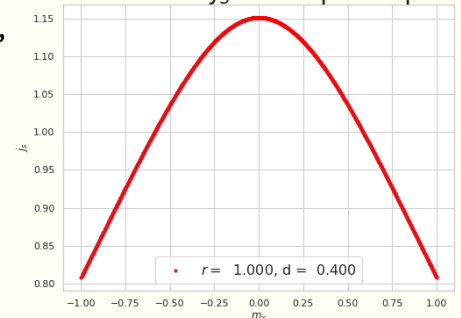
    fig = plt.figure(figsize=(8, 6))
    plt.scatter(arr_mx, arr_js, edgecolor="red",
               s=10, label=r'$r = $ %6.3f,
               d = %6.3f'%(r,d))
    plt.xlabel("$m_x$")
    plt.ylabel("$j_s$")

    plt.title("Зависимость $j_s$ от параметра $m_x$")
    plt.legend()

    plt.show()
```



Зависимость  $j_s$  от параметра  $m_x$



Поведение функций  $I_x, I_y$  при различных значениях параметров:

```
def funct_Ix(phi, mx, r, d):
    ''' Defines the integrand in the definition of current Ix,
        mx, r, d - parameters '''
    return (np.sin(phi) * np.exp(-d / np.cos(phi)) * np.sin(r*mx*np.tan(phi)))

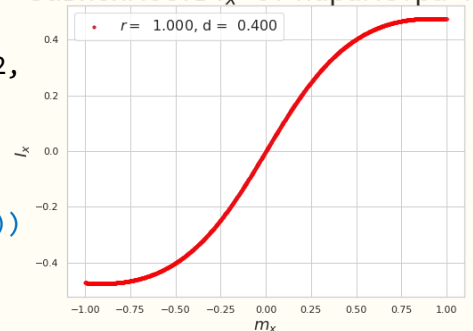
def funct_Iy(phi, mx, r, d):
    ''' Defines the integrand in the definition of current Iy,
        mx, r, d - parameters '''
    return (np.cos(phi) * np.exp(-d / np.cos(phi)) * np.cos(r*mx*np.tan(phi)))
```

```
@interact
def show_Ix(r=(0.1, 2.0, 0.1), d=(0.1, 0.8, 0.1)):
    Npoint = 1000 # количество точек
    arr_mx = np.linspace(-1, 1, Npoint, endpoint=True)
    arr_js = np.zeros(Npoint, dtype = np.float64)
    arr_err = np.zeros(Npoint, dtype = np.float64)
    for ind in range(Npoint):
        mx = arr_mx[ind]
        arr_js[ind], arr_err[ind] = quad(funcnt_Ix, -np.pi/2,
            np.pi/2,
            args=(mx,r,d))

    fig = plt.figure(figsize=(8, 6))
    plt.scatter(arr_mx, arr_js, edgecolor="red",
               s=10, label=r'$r = $ %6.3f, d = %6.3f'%(r,d))
    plt.xlabel("$m_x$")
    plt.ylabel("$I_x$")
    plt.title("Зависимость $I_x$ от параметра $m_x$")
    plt.legend()
    plt.show()
```



Зависимость  $I_x$  от параметра  $m_x$



**Вывод:** функция имеет вид кубической параболы (или многочлена нечетной степени)

## Подходы к ускорению вычислений

### Аппроксимация вычисленных интегралов для $m_x \in [-1, 1]$

Для аппроксимации интеграла воспользуемся модулем `scipy.optimize` библиотеки **SciPy**, функцией `curve_fit`

```
from scipy.optimize import curve_fit
```

```
# область изменения параметра mx
xnew = np.linspace(-1, 1, 2000, endpoint=True)

def func_P2(x, a, b, c):
    '''Defines a polynomial of the second degree'''
    return a*x*x + b*x + c
```

```
# значения параметров
```

```
d = 0.8
```

```
r = 1.1
```

```
# массивы
```

```
Npoint = 1000
```

```
arr_mx = np.linspace(-1, 1, Npoint, endpoint=True)
```

```
arr_js = np.zeros(Npoint, dtype = np.float64)
```

```
arr_err = np.zeros(Npoint, dtype = np.float64)
```

```
# вычисление интегралов
```

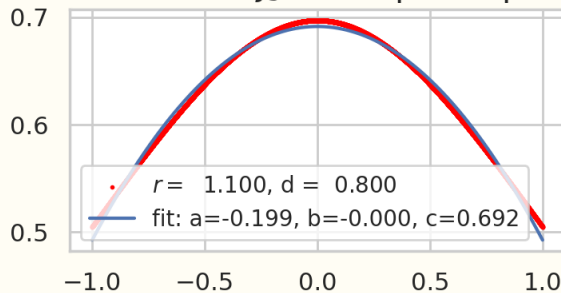
```
for ind in range(Npoint):
```

```
    mx = arr_mx[ind]
```

```
    arr_js[ind], arr_err[ind] = quad(func_js, -np.pi/2, np.pi/2, args=(mx,r,d))
```

```
popt, pcov = curve_fit(func_P2, arr_mx, arr_js)
```

Зависимость  $j_s$  от параметра  $m_x$



### Аппроксимация многочленом степени $n > 2$

Для решения задачи аппроксимации возможно применить различные подходы, например, реализованный в библиотеке **Numpy** `polyfit` или подход в библиотеке **SciPy**: `scipy.odr.polynomial`

```
from scipy import odr
```

```
poly_model = odr.polynomial(order) # factory function for a general polynomial model
```

### Многочлен второй степени $P_2$

```
from scipy import odr
```

```
# using the second order polynomial model
```

```
poly_model = odr.polynomial(2)
```

```
data = odr.Data(arr_mx, arr_js)
```

```
odr_obj = odr.ODR(data, poly_model)
```

```
output = odr_obj.run() # running ODR fitting
```

```
poly = np.poly1d(output.beta[:, -1])
```

```
poly_y = poly(arr_mx)
```

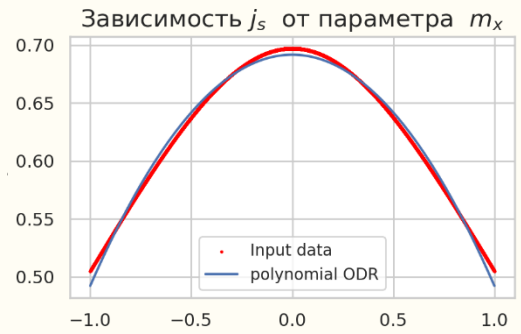
В классе `numpy.poly1d`:  
`class numpy.poly1d(c_or_r, r=False, variable=None)[source]` класс одномерных полиномов.

Коэффициенты полинома представлены по убывающим степеням или, если значение второго параметра равно `True`, корни полинома (значения, при которых значение полинома равно 0). Например, `poly1d([1,2,3])` возвращает объект, представляющий  $x^2 + 2x + 3$ .

```
plt.figure(figsize=(5, 3), dpi=200)
plt.scatter(arr_mx, arr_js, edgecolor="red",
            s=1, label="Input data")
plt.plot(arr_mx, poly_y, label="polynomial ODR")
plt.xlabel("$m_x$")
plt.ylabel("$j_{s}$")

plt.title("Зависимость $j_{s}$ от параметра $m_x$")
plt.legend()

plt.show()
```

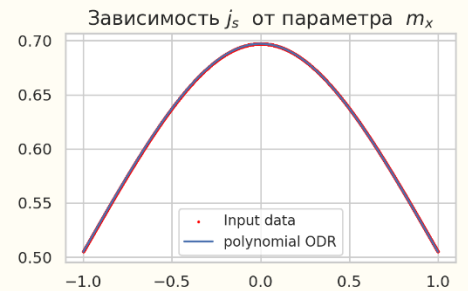


Многочлен восьмой степени  $P_8$

```
# using the 8th order polynomial model
poly_model = odr.polynomial(8)
data = odr.Data(arr_mx, arr_js)
odr_obj = odr.ODR(data, poly_model)

output = odr_obj.run() # running ODR fitting

poly = np.poly1d(output.beta[::-1])
poly_y = poly(arr_mx)
```



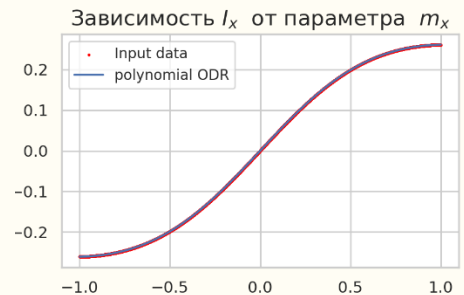
**Вывод:** далее будем аппроксимировать интеграл  $I_y$  многочленами 8-ой степени.

Аналогично для интеграла  $I_x$ :

```
# using the 9th order polynomial model
poly_model = odr.polynomial(9)
data = odr.Data(arr_mx, arr_js)
odr_obj = odr.ODR(data, poly_model)

output = odr_obj.run()

poly = np.poly1d(output.beta[::-1])
poly_y = poly(arr_mx)
```



**Вывод:** далее будем аппроксимировать интеграл  $I_x$  многочленом 9-ой степени.

## Численное решение задачи Коши: библиотека SciPy

**Пример 1.** Численно решить задачу Коши

$$\begin{cases} dy/dt = y \cos(t), \\ y(0) = y_0, \end{cases}$$

имеющее аналитическое решение

$$y_{exact} = y_0 e^{\sin(t)}.$$

• Определяем правую часть уравнения

```
def F_right(t, y):
    ''' Определяет правую часть ДУ,
        примера 1'''
    return y*np.cos(t)
```

• Определяем параметры численного счета

```
t0 = 0
tf = 10
nt = 1000
# Массив точек, в которых будет находится решение
t_e = np.linspace(t0, tf, nt)
# Начальное условие
y0 = np.array([3])
```

• Функция библиотеки **SciPy** для решения начальной задачи

```
sol_1 = solve_ivp(F_right, [t0, tf], y0, t_eval=t_e, method='RK45', rtol=1e-9, atol=1e-8)
```

`F_right` – правая часть дифференциального уравнения;  
`[t0, tf]` – отрезок интегрирования;  
`y0` – начальное условие;  
`t_eval` – точки сетки, в которых следует вычислить решение;  
`method` – метод интегрирования;  
`rtol, atol` – относительная и абсолютная погрешности.

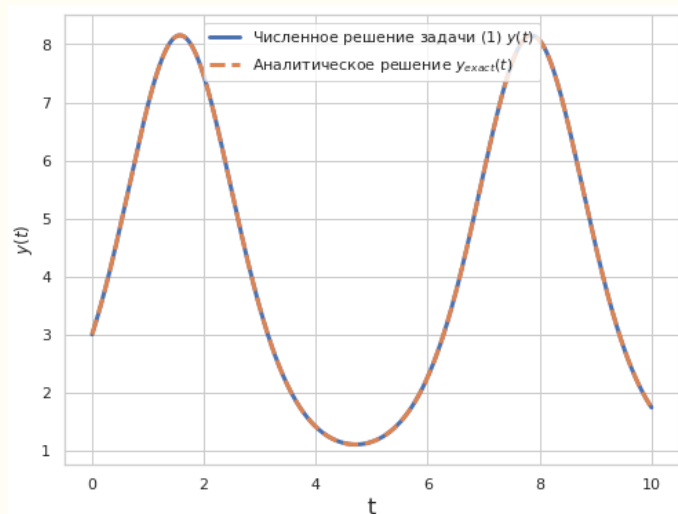


Рис. 1. График численного и аналитического решения

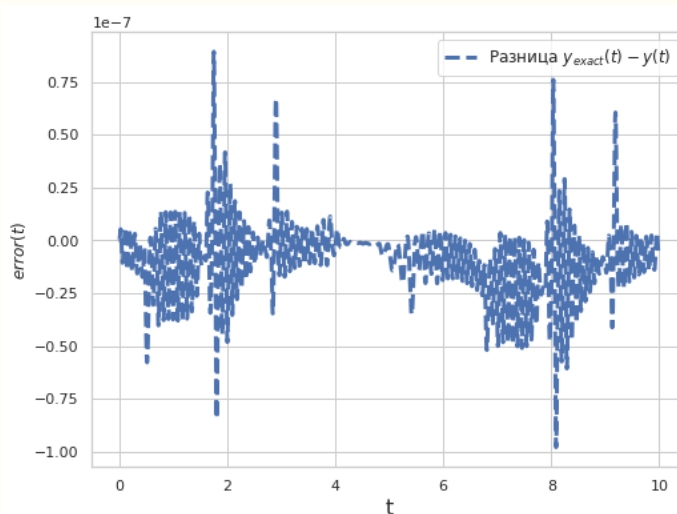


Рис. 2. Разница между аналитическим и численным решением

**Пример 2.** Численно решить задачу Коши

$$\begin{cases} dy/dt = y \cos(\omega t), \\ y(0) = y_0, \end{cases}$$

$\omega$  – параметр

имеющее аналитическое решение

$$y_{exact} = \begin{cases} y_0 e^{\frac{1}{\omega} \sin(\omega t)} & \text{при } \omega \neq 0, \\ y_0 e^t & \text{при } \omega = 0. \end{cases}$$

Отметим, что в модель входит параметр  $\omega$ .

- Определяем правую часть уравнения
 

```
def F_right2(t, y, omega):
    ''' Определяет правую часть ДУ,
        примера 2,
        omega - параметр'''
    return y*np.cos(omega*t)
```

- Определяем параметры модели и численного счета
 

```
omega = np.pi/2
# Параметры численного счета
t0 = 0
tf = 10
nt = 1000
# Массив точек, в которых будет находится решение
t_e = np.linspace(t0, tf, nt)
# Начальное условие
y0 = np.array([3])
```

- Для корректной передачи параметра в функцию SciPy воспользуемся функцией **partial** из модуля **functools**, которая *частично* применяет аргументы к вызываемой функции.

```
f = partial(F_right2, omega=omega)
t_e = np.linspace(t0, tf, nt)
sol_2 = solve_ivp(f, [t0, tf], y0, t_eval=t_e, method='RK45', rtol=1e-9, atol=1e-8)
```

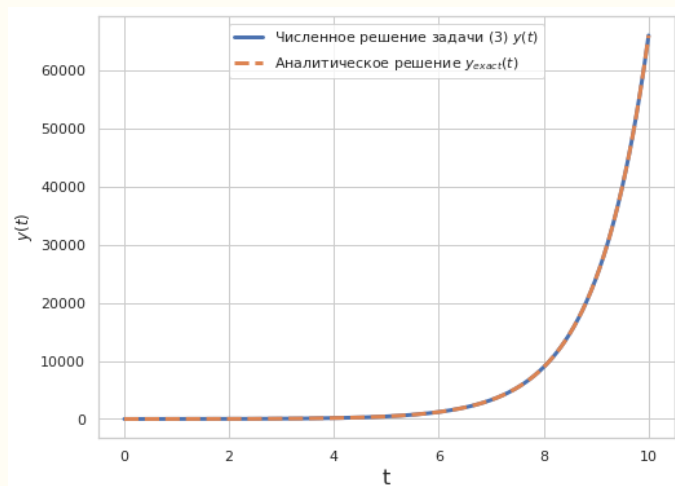


Рис. 3. График численного и аналитического решения примера 2 при  $\omega = 0$

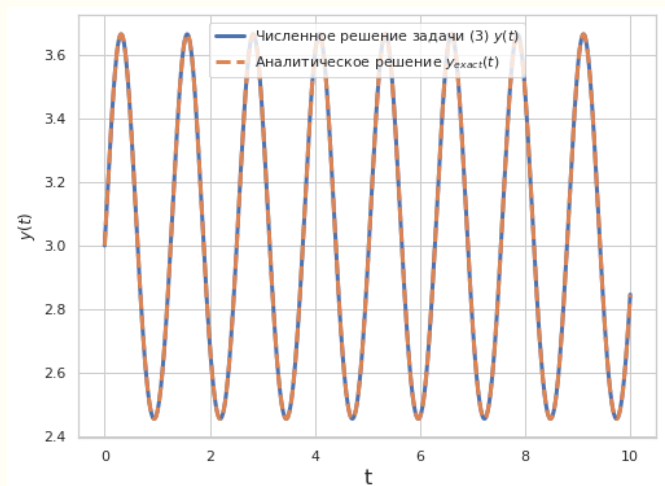


Рис. 4. График численного и аналитического решения примера 2 при  $\omega = 5$



# Математическое моделирование динамики джозефсоновского переход сверхпроводник/ферромагнетик/сверхпроводник на поверхности трехмерного топологического изолятора

## I. Основные понятия

**Джозефсоновский переход** — это связь двух сверхпроводящих слоев посредством тонкого слоя несверхпроводящего барьера, в котором при пропускании электрического тока в зависимости от его величины наблюдается *стационарный и нестационарный эффект Джозефсона*.

**Стационарный эффект Джозефсона.** При пропускании тока  $j$  ниже критического значения  $j_c$  ( $j < j_c$ ) в джозефсоновском переходе отсутствует напряжение ( $V = 0$ ) и через переход течет сверхпроводящий ток  $j_s$ . Данный ток пропорционален синусу разности фаз  $\varphi$  параметров порядка (волновой функции или функции состояния) сверхпроводящих слоев

$$j_s = j_c \sin \varphi. \quad (1)$$

Это выражение называется *ток-фазовое соотношение* джозефсоновского перехода.

**Нестационарный эффект Джозефсона.** При увеличении тока  $j$  выше критического значения  $j_c$  ( $j > j_c$ ) возникает переменное напряжение  $V$  в переходе и оно пропорционально производной разности фаз по времени  $t$

$$V = \frac{\hbar}{2e} \frac{d\varphi}{dt}, \quad (2)$$

где  $\hbar$  – постоянная планка,  $e$  – заряд электрона.

**Фи-0 Джозефсоновский переход.** Если в качестве несверхпроводящего барьера использовать ферромагнитный слой с спинорбитальным взаимодействием, тогда на ток-фазовом соотношении (сверхпроводящем токе) возникает фазовый сдвиг  $\varphi_0$  зависящий от компоненты намагниченности.

$$j_s = j_c \sin(\varphi - \varphi_0). \quad (3)$$

Такие переходы называются *Фи-0 джозефсоновскими переходами*.

## II. Теоретическая модель и система уравнений

Рассмотрим S/F/S структуру, где два обычных s-волновых сверхпроводника и ферромагнетик, нанесенные на поверхность 3D TI, образуют джозефсоновский переход. Схематический вид такого перехода представлен на Рис.1.

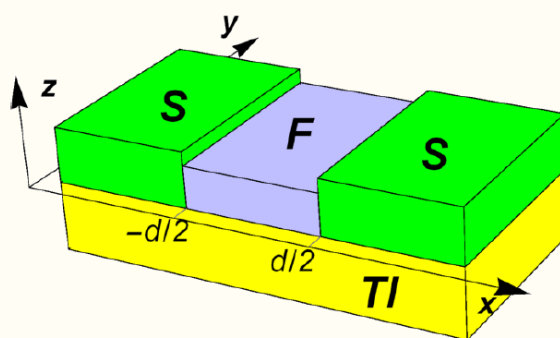


Рис. 1. Схема S/F/S джозефсоновского перехода на поверхности 3D топологического изолятора. Легкая ось ферромагнетика направлена вдоль оси  $y$

Ток-фазовое соотношение этого перехода задается выражением

$$j_s = j_c \sin(\varphi - \varphi_0), \quad (4)$$

где  $j_c$  – критический ток, зависящий от  $x$ -компоненты намагниченности,  $\varphi$  – джозефсоновская разность фаз,  $\varphi_0 = rm_y$  – аномальный сдвиг фазы,  $m_y = M_y/M_S$  –  $y$  компонента намагниченности ( $M_y$ ) нормированная на намагниченность насыщения  $M_S$ ,  $r = 2dh_{\text{exc}}/v_F$  – безразмерный параметр, определяющий величину спин-орбитального взаимодействия,  $d$  – толщина ферромагнитного барьера,  $h_{\text{exc}}$  – обменное поле,  $v_F$  – скорость Ферми.

Отличительной чертой рассматриваемого джозефсоновского перехода является то, что критический ток сильно зависит от ориентации намагниченности, а именно, от  $x$ -компоненты намагниченности<sup>1</sup> в плоскости вдоль направления тока и задается выражением

$$j_c = j_b \int_{-\pi/2}^{\pi/2} \cos \phi \exp\left(-\frac{\tilde{d}}{\cos \phi}\right) \cos(rm_x t g \phi) d\phi, \quad (5)$$

где  $\phi$  – угол между направлением квазичастичного тока и осью  $x$ ,  $m_x = M_x/M_S$  –  $x$ -компонента намагниченности нормированная на  $M_S$ ,  $j_b = \frac{ev_F N_F \Delta^2}{\pi^2 T}$ ,  $\Delta$  – сверхпроводящий параметр порядка,  $T$  – температура,  $N_F$  – концентрация частиц вблизи уровня Ферми,  $v_F$  – скорость Ферми,  $\tilde{d}$  – безразмерная длина контакта.

Динамика вектора намагниченности ( $\mathbf{M}$ ) ферромагнитного слоя описывается в рамках уравнения Ландау - Лифшица - Гильберта (ЛЛГ):

$$\frac{d\mathbf{M}}{dt} = -\gamma \mathbf{M} \times \mathbf{H}_{\text{eff}} + \frac{\alpha}{M_S} \mathbf{M} \times \frac{d\mathbf{M}}{dt}, \quad (6)$$

где  $\gamma$  – гиромагнитное отношение,  $\alpha$  – гильбертовское затухание и  $\mathbf{H}_{\text{eff}}$  – эффективное поле. Эффективное поле определяется варьированием полной энергии системы по вектору намагниченности

$$\mathbf{H}_{\text{eff}} = -\frac{1}{V_F} \frac{\delta E_t}{\delta \mathbf{M}}, \quad (7)$$

где  $V_F$  – объем ферромагнитного слоя. Полная энергия системы состоит из энергии магнитной анизотропии

$$E_M = -\frac{KV_F}{2} \left(\frac{M_y}{M_S}\right)^2, \quad (8)$$

где  $K$  – константа анизотропии, и джозефсоновской энергии

$$E_J = \frac{\Phi_0 j_c S}{2\pi} [1 - \cos(\varphi - rm_y)], \quad (9)$$

где  $\Phi_0$  – квант магнитного потока,  $S$  – площадь перехода. Таким образом, компоненты эффективного поля в нормированных единицах могут быть записаны в виде:

$$\begin{aligned} h_x &= \frac{H_{\text{eff},x}}{H_F} = \frac{GrI_x}{j_{c0}} [1 - \cos(\varphi - rm_y)], \\ h_y &= \frac{H_{\text{eff},y}}{H_F} = \frac{GrI_y}{j_{c0}} \sin(\varphi - rm_y) + m_y, \\ h_z &= \frac{H_{\text{eff},z}}{H_F} = 0. \end{aligned} \quad (10)$$

<sup>1</sup> M. Nashaat, I. V. Bobkova, A. M. Bobkov, Yu. M. Shukrinov, I. R. Rahmonov, and K. Sengupta, Phys. Rev. B **100** 054506 (2019).

где  $G = \Phi_0 j_b S / 2\pi K V_F$  – отношение амплитуды джозефсоновской энергии к магнитной,  $H_F = \omega_F / \gamma = K / M_S$ ,  $\omega_F$  – собственная частота ферромагнитного резонанса, а  $I_x$  и  $I_y$  интегральные выражения определяемые как:

$$\begin{aligned} I_x &= \int_{-\pi/2}^{\pi/2} \sin \phi \exp\left(-\frac{\tilde{d}}{\cos \phi}\right) \sin(rm_x tg \phi) d\phi, \\ I_y &= \int_{-\pi/2}^{\pi/2} \cos \phi \exp\left(-\frac{\tilde{d}}{\cos \phi}\right) \cos(rm_x tg \phi) d\phi. \end{aligned} \quad (11)$$

Здесь  $j_{c0}$  определяет выражение для критического тока при  $m_x = 0$  и записывается как

$$j_{c0} = \int_{-\pi/2}^{\pi/2} \cos \phi \exp\left(-\frac{\tilde{d}}{\cos \phi}\right) d\phi. \quad (12)$$

Таким образом, в нормированных величинах, получим систему уравнений:

$$\begin{aligned} \frac{dm_x}{dt} &= -\frac{\omega_F}{1 + \alpha^2} \left( (m_y h_z - m_z h_y) + \alpha [m_x (m_x h_x + m_y h_y + m_z h_z) - h_x m^2] \right), \\ \frac{dm_y}{dt} &= -\frac{\omega_F}{1 + \alpha^2} \left( (m_z h_x - m_x h_z) + \alpha [m_y (m_x h_x + m_y h_y + m_z h_z) - h_y m^2] \right), \\ \frac{dm_z}{dt} &= -\frac{\omega_F}{1 + \alpha^2} \left( (m_x h_y - m_y h_x) + \alpha [m_z (m_x h_x + m_y h_y + m_z h_z) - h_z m^2] \right). \end{aligned} \quad (13)$$

При заданном значении напряжении можно считать разность фаз  $\phi$  линейной функцией от времени, т.е.

$\phi = Vt$ . Тогда выражения для компонент эффективного поля записываются в виде

$$\begin{aligned} h_x &= \frac{GrI_x}{j_{c0}} [1 - \cos(Vt - rm_y)], \\ h_y &= \frac{GrI_y}{j_{c0}} \sin(Vt - rm_y) + m_y, \\ h_z &= 0. \end{aligned} \quad (14)$$

### III. Постановка задачи

В нашем случае легкая ось намагниченности в ферромагнитном слое направлена вдоль оси  $y$ , т.е. направления  $m_y = \pm 1$  должны быть стабильными. Однако из-за особенности модели при определенных значениях параметров реализуются четырехкратно вырожденные стабильные состояния намагниченности.

Наша задача заключается в том, чтобы на основе математического моделирования продемонстрировать реализацию этих вырожденных состояний.

### IV. Ожидаемые результаты

Решая численно задачу Коши для системы дифференциальных уравнений (13), можно показать эти четырехкратно вырожденные стабильные состояния:

1.  $m_x > 0, m_y > 0$ ;
2.  $m_x < 0, m_y > 0$ ;
3.  $m_x > 0, m_y < 0$ ;
4.  $m_x < 0, m_y < 0$ .

На Рис.2 представлены временные зависимости компонент намагниченности рассчитанные при различных начальных условиях: (а)  $m_x(0) = 0.5$ ,  $m_y(0) > 0$ ,  $m_z(0) = 0$ ; (б)  $m_x(0) = -0.5$ ,  $m_y(0) > 0$ ,  $m_z(0) = 0$ ; (с)  $m_x(0) = -0.5$ ,  $m_y(0) < 0$ ,  $m_z(0) = 0$ ; (д)  $m_x(0) = 0.5$ ,  $m_y(0) < 0$ ,  $m_z(0) = 0$ .

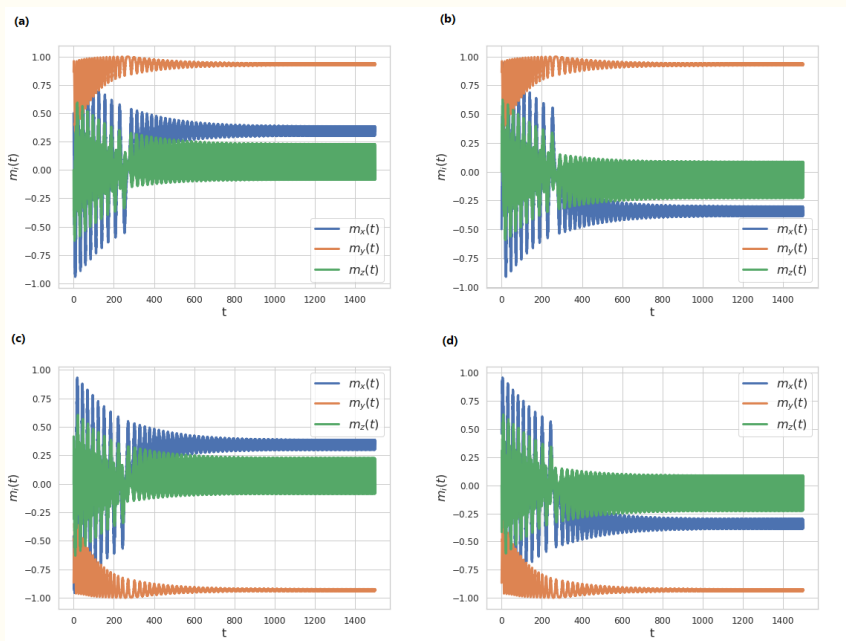


Рис. 2. Временные зависимости компонент намагниченности, рассчитанные при напряжении  $V = 5$  демонстрирующие возможные стабильные состояния

На этих рисунках видно вышеупомянутые возможные стабильные состояния: а) состояния 1; б) состояния 2; с) состояния 3; д) состояния 4.

### Ускорение вычислений с использованием библиотеки Joblib

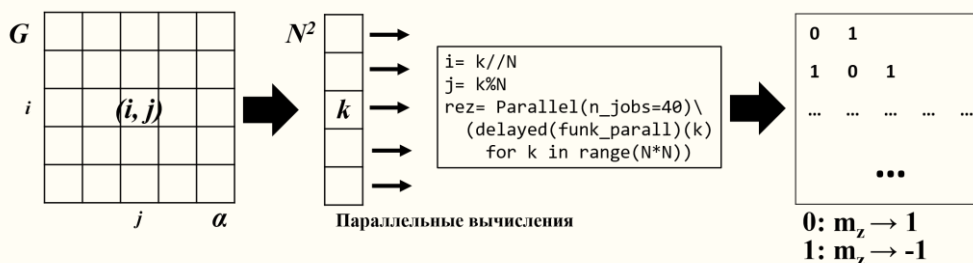


Рис. 3. Схема распараллеливания задачи с применением библиотеки Joblib

Для ускорений вычислений при моделировании переворота намагниченности в плоскости параметров  $(G, \alpha)$  или  $(G, r)$  можно использовать библиотеку Joblib.

<pre># подключение библиотеки Joblib import joblib from joblib import Parallel, delayed</pre>	<pre># доступное количество CPU потоков print(f"Number of CPU: {joblib.cpu_count()}")</pre> <p>Out: Number of CPU: 40</p>
<pre>rez = Parallel(n_jobs=10)(delayed(funk_parall)(k) for k in range(N * N))</pre> <p><b>n_jobs</b> – используемое количество потоков. Так же мы можем передать значение -1 для использования всех ядер или -2 для использования всех ядер, кроме одного.          Функция <b>delayed</b> используется для отсрочки выполнения кода. Она используется для того, чтобы библиотека сформировала список вызова функций, которые нужно выполнить параллельно. Этот список затем передается в функцию <b>Parallel</b>, которая занимается параллельным выполнением задач.  <b>funk_parall</b> – функция, вычисления в которой необходимо ускорить;  <b>k</b> – входной параметр функции, по которому будет выполняться распараллеливание.</p>	