Практическое занятие
Инструментарий на основе Python-библиотек и экосистемы Jupyter для решения научных и прикладных задач

Башашин М.В., Бежанян Т.Ж., Бутенко Ю.А., Воронцов А.С., Зуев М.И., Киракосян М.Х., Матвеев М.А., Нечаевский А.В., Пряхина Д.И., Рахмонов И.Р., Рахмонова А.Р., Стрельцова О.И.

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

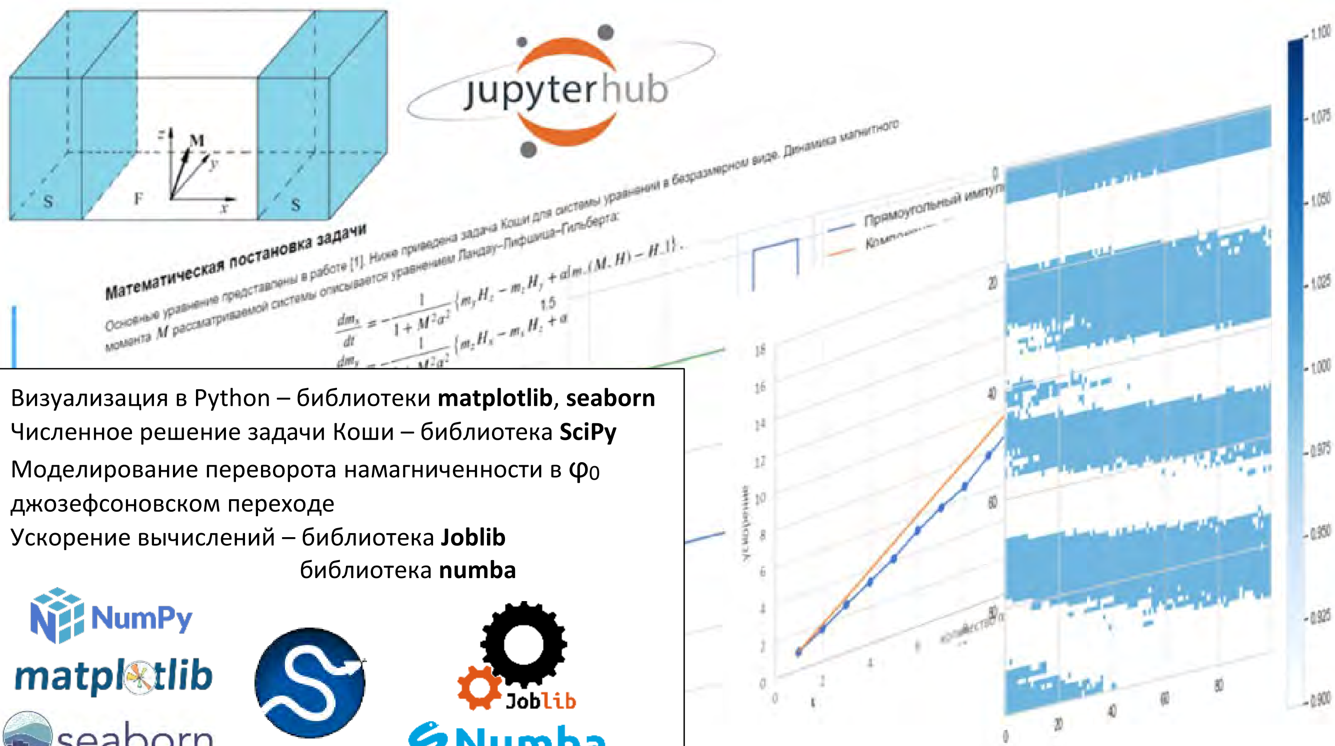
HPClab component
VM with JupyterHub and SLURM [**https://jlabhpc.jinr.ru**]
- Intel Xeon Gold 6126 (24 Cores @ 2.6 GHz)
- 32 GB RAM

Educational component
JupyterLab Server [**https://studhub.jinr.ru**]
[**https://studhub2.jinr.ru**]
- 2x Intel Xeon Gold 6152 (22 Cores @ 2.1 GHz)
- 512 GB RAM

Computation component
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

[**https://jhub3.jinr.ru**]
- 2x Intel Xeon E5 2698v4 (20 Cores @ 2.2 GHz)
- 8x NVIDIA Tesla V100 SXM2 16 GB HBM2
- 512 GB RAM

План занятия



- Визуализация в Python – библиотеки **matplotlib**, **seaborn**
- Численное решение задачи Коши – библиотека **SciPy**
- Моделирование переворота намагниченности в φ_0 джозефсоновском переходе
- Ускорение вычислений – библиотека **Joblib**
  библиотека **numba**

Создание функций

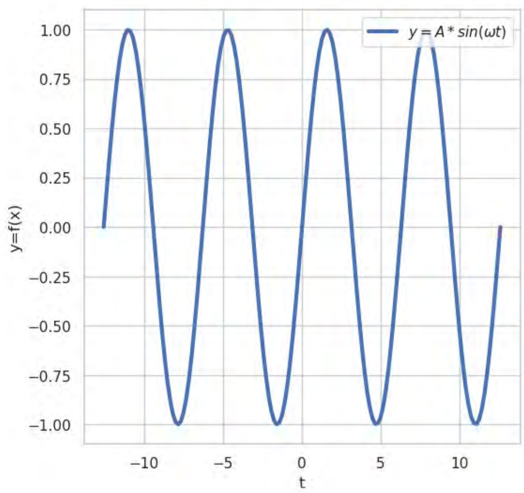
```
def f_sin(t, A, omega):
    ''' Определяет значение функции A*sin(omega*t),
        A, omega - параметры'''
    return (A*np.sin(omega*t))
```

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

```
import numpy as np # подключение библиотеки
a = np.array([1, 2, 3]) # создание одномерного массива
x = np.arange(0, 30, 2) # создание одномерного массива с числами от 0 до 30 с шагом 2
x = np.linspace(0, 2*np.pi, 100, endpoint=True) # создание одномерного массива с числами от 0
до 2π с количеством точек 100
```

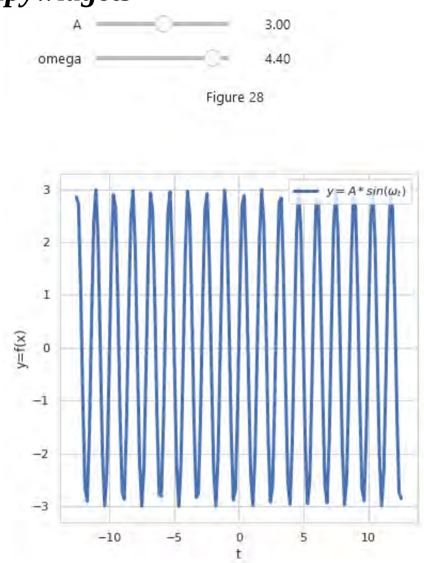
Построение графиков – библиотеки matplotlib, seaborn

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

# подключение библиотеки seaborn
import seaborn as sns
sns.set()
sns.set(style="whitegrid")

t = np.linspace(-4*np.pi, 4*np.pi, 150, endpoint=True)

# размеры графика
fig = plt.figure(figsize=(6, 6))
# отрисовка графика по координатам x и y
plt.plot(t, f_sin(t, A, omega),
label='$y=A*sin(\\omega t )$', linewidth=3.0)
plt.xlabel('x', size=12) # подпись оси x
plt.ylabel('y', size=12) # подпись оси y
plt.legend(loc='upper right') # добавление легенды
plt.show() # отображение графика
```



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_sin(A=(1.0, 5.0, 1.0), omega=(0.1, 5.0, 0.1)):
    t = np.linspace(-4*np.pi, 4*np.pi, 150, endpoint=True)
    fig = plt.figure(figsize=(6, 6))
    plt.plot(t, f_sin(t, A, omega),
             label='$y=A*sin(\\omega_t )$', linewidth=3.0)
    plt.xlabel('t')
    plt.ylabel('y=f(x)')
    plt.legend(loc='upper right')
    plt.show()
```



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

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

- Определяем параметры численного счета

```python
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

```python
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.

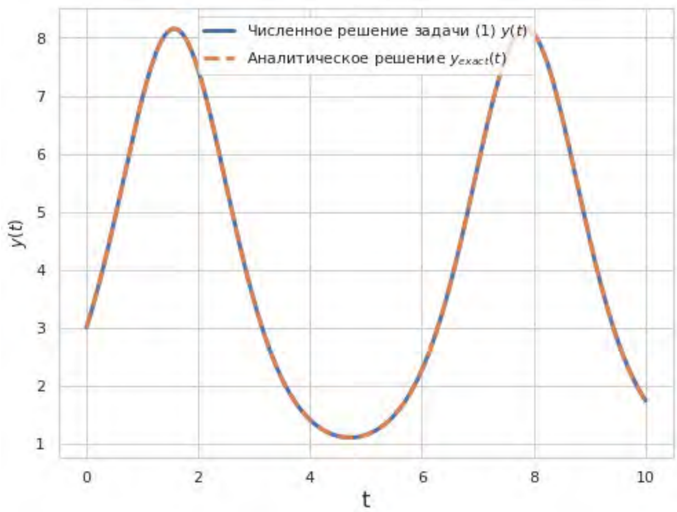


Рис. 1. График численного и аналитического решения

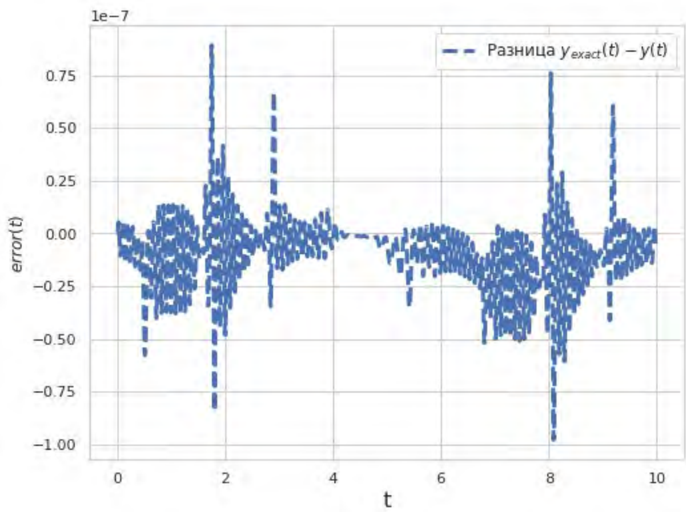


Рис. 2. Разница между аналитическим и численным решением

Пример 2. Численно решить задачу Коши

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

ω – 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 ω.

- 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)
```

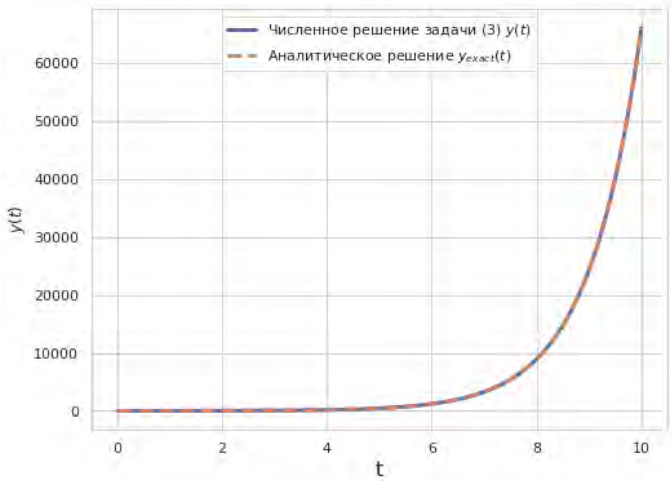


Рис. 3. График численного и аналитического решения примера 2 при ω = 0

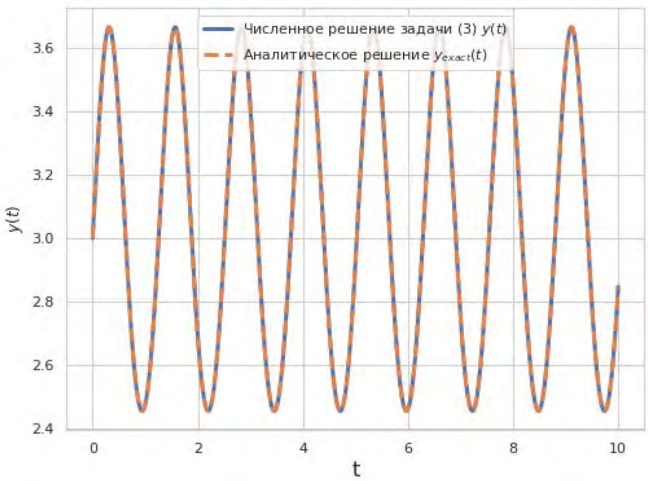


Рис. 4. График численного и аналитического решения примера 2 при ω = 5

Моделирование переворота намагниченности в φ_0 джозефсоновском переходе

В качестве примера рассмотрим задачу о реализации переворота намагниченности в так называемом φ_0 джозефсоновском переходе под воздействием импульса тока и вычислении периодичности появления интервалов переворота намагниченности с изменением параметров модели.

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 I ниже критического значения I_c $\left(I < I_c\right)$ в джозефсоновском переходе отсутствует напряжение $\left(V = 0\right)$ и через переход течет сверхпроводящий ток I_s. Данный ток пропорционален синусу разности фаз φ параметров порядка (волновой функции или функции состояния) сверхпроводящих слоев

$$I_s = I_c \sin\varphi \tag{1}$$

Это выражение называется ток фазовое соотношение джозефсоновского перехода.

Нестационарный эффект Джозефсона. При увеличении тока I выше критического значения I_c $\left(I > I_c\right)$ возникает переменное напряжение V в переходе, и оно пропорционально производной разности фаз по времени t

$$V = \frac{\hbar}{2e}\frac{d\varphi}{dt}, \tag{2}$$

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

φ_0 джозефсоновский переход. Если в качестве несверхпроводящего барьера использовать ферромагнитный слой со спинорбитальным взаимодействием и с нарушением симметрии относительно обращения времени, то на токфазовом соотношении (сверхпроводящем токе) возникает фазовый сдвиг φ_0.

$$I_s = I_c \sin\left(\varphi - \varphi_0\right), \tag{3}$$

где φ_0 зависит от намагниченности. Такой переход называется *φ_0 джозефсоновским переходом*.

Система уравнений

В φ_0 джозефсоновском переходе (Рис. 4) динамическими переменными являются разность фаз φ сверхпроводящих слоев и вектор намагниченности **M** ферромагнитного слоя. Согласно резистивной модели, уравнение для разности фаз пишется как сумма сверхпроводящего и квазичастичного (одноэлектронного) токов через переход.

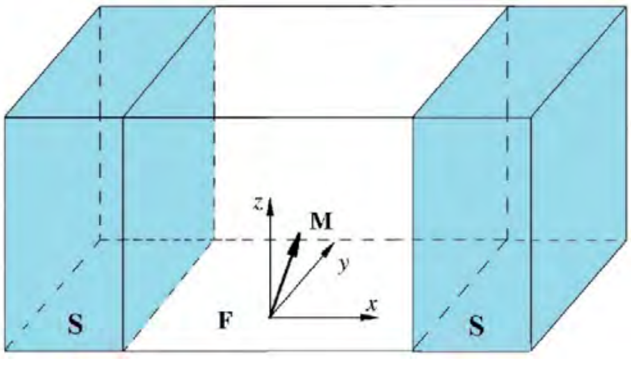


Рис. 5. Схематический вид φ_0 – джозефсоновского перехода; S и F – сверхпроводниковые и ферромагнитный слои, соответственно, **M** *– вектор намагниченности ферромагнитного слоя, легкая ось которого направлена вдоль оси z. Импульс внешнего тока направлен вдоль оси x.*

$$I=\frac{\hbar}{2eR}\left(\frac{d\varphi}{dt}-\frac{d\varphi_0}{dt}\right)+I_c\sin\left(\varphi-\varphi_0\right), \tag{4}$$

где R – сопротивление джозефсоновского перехода.
Динамика вектора намагниченности описывается уравнением Ландау-Лифщица-Гильберта

$$\frac{d\mathbf{M}}{dt}=-\gamma\left[\mathbf{M}\times\mathbf{H_{eff}}\right]+\frac{\alpha}{M_0}\left[\mathbf{M}\times\frac{d\mathbf{M}}{dt}\right], \tag{5}$$

где γ – гиромагнитное отношение, α – гильбертовское затухание, M_0 – величина насыщения намагниченности или модуль вектора $\mathbf{M}$. Здесь $\mathbf{H_{eff}}$ обозначает вектор эффективного магнитного поля, которое определяется выражением

$$\mathbf{H_{eff}}=\frac{K}{M_0}\left[Gr\sin\left(\varphi-r\frac{M_y}{M_0}\right)\mathbf{e_y}+\frac{M_z}{M_0}\mathbf{e_z}\right], \tag{6}$$

где G – отношение энергии Джозефсона к энергии магнитной анизотропии, r – параметр спин-орбитального взаимодействия, $\mathbf{e_y}$, $\mathbf{e_z}$ – единичные вектора. Здесь учтено, что $\varphi_0=rM_y/M_0$.

Разрешая уравнение (1) относительно $d\mathbf{M}/dt$ и учитывая уравнение для разности фаз (4), можно записать замкнутую систему уравнений

$$\begin{cases}\dfrac{d\mathbf{M}}{dt}=-\Omega_F\left[1+\left(\alpha\dfrac{\mathbf{M}}{M_0}\right)^2\right]^{-1}\left\{\dfrac{M_0}{K}\left[\mathbf{M}\times\mathbf{H_{eff}}\right]+\dfrac{\alpha}{K}\left[\mathbf{M}\left(\mathbf{M}\mathbf{H_{eff}}\right)-\mathbf{H_{eff}}\mathbf{M}^2\right]\right\},\\ \dfrac{d\varphi}{dt}=\dfrac{r}{M_0}\dfrac{dM_y}{dt}+\dfrac{2eR}{\hbar}\left[I-I_c\sin\left(\varphi-r\dfrac{M_y}{M_0}\right)\right],\end{cases} \tag{7}$$

учитывая, что $\mathbf{m}=\mathbf{M}/M_0$ (где $\mathbf{m}^2=1$), $\mathbf{h_{eff}}=M_0\mathbf{H_{eff}}/K$, $\omega_F=\Omega_F/\omega_c$, где $\omega_c=\left(2eI_cR\right)/\hbar$ и, нормируя t на ω_c^{-1}, I на I_c получим систему уравнений в безразмерных величинах.

$$\begin{cases}\dfrac{d\mathbf{m}}{dt}=-\dfrac{\omega_F}{1+\alpha^2}\left(\left[\mathbf{m}\times\mathbf{h_{eff}}\right]+\alpha\left[\mathbf{m}\left(\mathbf{m}\mathbf{h_{eff}}\right)-\mathbf{h_{eff}}\right]\right),\\ \dfrac{d\varphi}{dt}=r\dfrac{dm_y}{dt}+I-\sin\left(\varphi-rm_y\right),\end{cases} \tag{8}$$

где компоненты вектора $\mathbf{h_{eff}}$ определяются выражениями $h_x=0$, $h_y=rG\sin\left(\varphi-rm_y\right)$, $h_z=m_z$.

А в скалярном виде система уравнений записывается

$$\begin{cases}\dfrac{dm_x}{dt}=-\dfrac{\omega_F}{1+\alpha^2}\left\{\left(m_yh_z-m_zh_y\right)+\alpha\left[m_x\left(m_xh_x+m_yh_y+m_zh_z\right)-h_x\right]\right\},\\ \dfrac{dm_y}{dt}=-\dfrac{\omega_F}{1+\alpha^2}\left\{\left(m_zh_x-m_xh_z\right)+\alpha\left[m_y\left(m_xh_x+m_yh_y+m_zh_z\right)-h_y\right]\right\},\\ \dfrac{dm_z}{dt}=-\dfrac{\omega_F}{1+\alpha^2}\left\{\left(m_xh_y-m_yh_x\right)+\alpha\left[m_z\left(m_xh_x+m_yh_y+m_zh_z\right)-h_y\right]\right\},\\ \dfrac{d\varphi}{dt}=r\dfrac{dm_y}{dt}+I-\sin\left(\varphi-rm_y\right),\end{cases} \tag{9}$$

Начальное условие для этой системы задается в виде $m_x=0$, $m_y=0$, $m_z=1$, $\varphi=0$.

Внешний ток задается в виде прямоугольного импульса

$$I=I\left(t\right)=\begin{cases}A_s,\ t\in\left[t_s-\Delta t/2,t_s+\Delta t/2\right],\\ 0,\ otherwise.\end{cases} \tag{10}$$

где A_s – амплитуда и Δt продолжительность импульса тока, t_s – середина импульса.

Параметры модели: G – отношение энергии Джозефсона к энергии магнитной анизотропии; r – константа спин-орбитального взаимодействия; α – параметр диссипации Гильберта; ω_F – частота ферромагнитного резонанса.

Начальные условия предполагают, что все компоненты магнитного момента, кроме m_z, равны нулю:

$$\boxed{\text{IC: } m_x(0)=0,\ m_y(0)=0,\ m_z(0)=1,\ \phi(0)=0.}$$

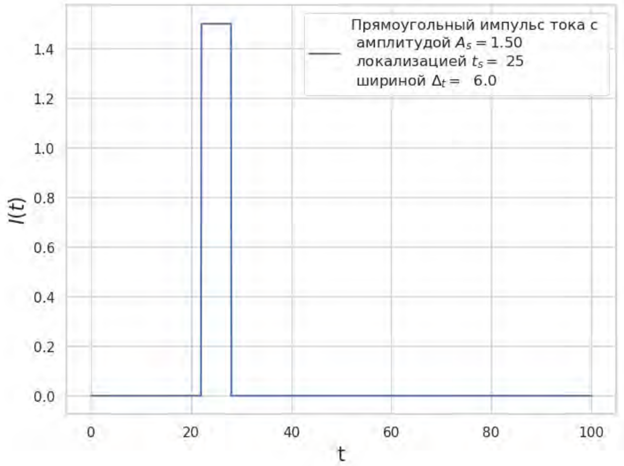


Рис. 6. Прямоугольный импульс тока

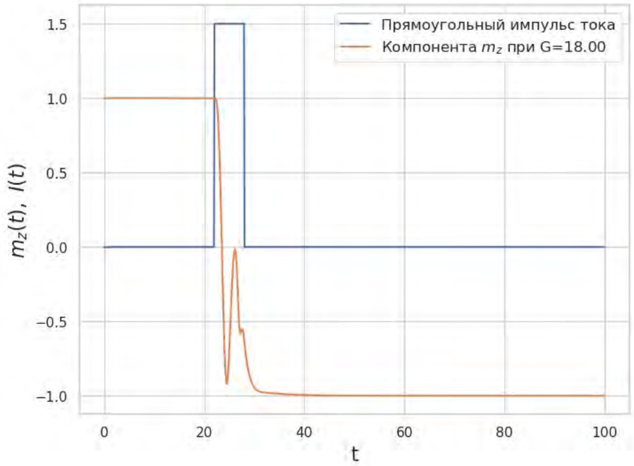


Рис. 7. Импульс тока и график, показывающий переворот компоненты m_z магнитного момента

Ускорение вычислений с использованием библиотеки Joblib

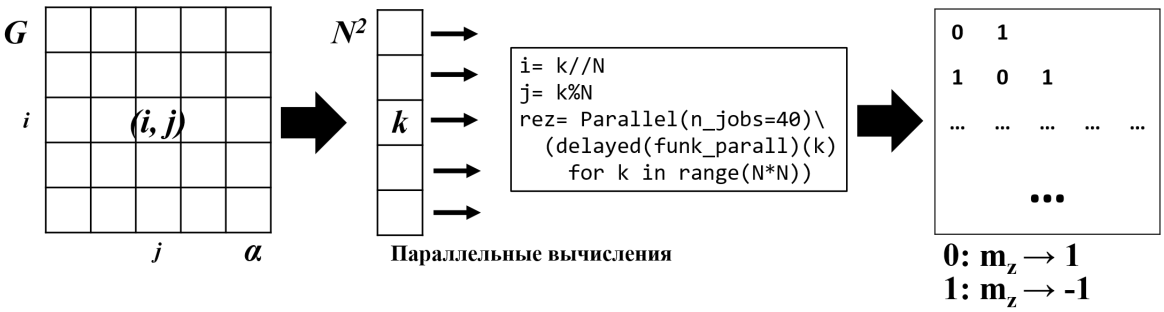


Рис. 8. Схема распараллеливания задачи с применением библиотеки Joblib

Для ускорений вычислений при моделировании переворота намагничености в плоскости параметров (G,α) или (G,r) можно использовать библиотеку Joblib.

```
# подключение библиотеки Joblib
import joblib
from joblib import Parallel, delayed
```

```
# доступное количество CPU потоков
print(f"Number of CPU: {joblib.cpu_count()}")
Out: Number of CPU: 80
```

```
rez = Parallel(n_jobs=10)(delayed(funk_parall)(k) for k in range(N * N))
```

- `n_jobs` – используемое количество потоков. Так же мы можем передать значение **-1** для использования всех ядер или **-2** для использования всех ядер, кроме одного.
- Функция **`delayed`** 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 **Parallel**, которая занимается параллельным выполнением задач.
- **`funk_parall`** – функция, вычисления в которой необходимо ускорить;
- `k` – входной параметр функции, по которому будет выполняться распараллеливание.

Периодичность появления интервалов переворота намагниченности в φ_0 джозефсоновском переходе под воздействием импульса тока

Ресурсоемкие вычисления при исследовании периодичности интервалов переворотов:

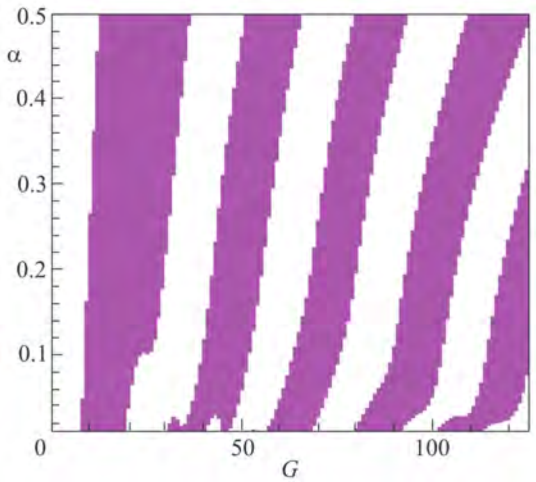


Рис. 9. Демонстрация периодичности интервалов переворота в плоскости (G,α). Результаты получены с шагом $\Delta G = 1$ и $\Delta\alpha = 0.001$ при $A_s = 1.5$, $r = 0.1$, $t_0 = 25$, $\Delta t = 6$, $\omega_F = 1$

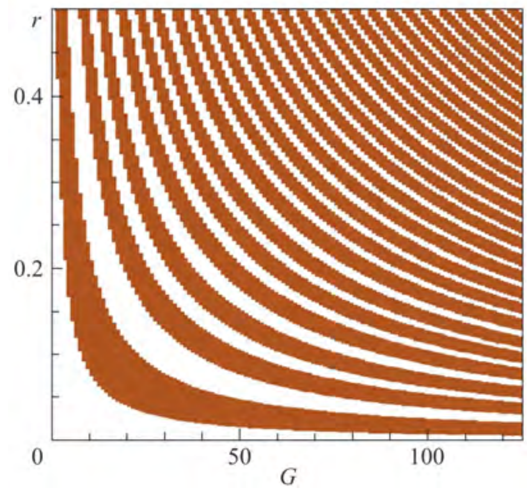


Рис. 10. Демонстрация периодичности интервалов переворота в плоскости (G,r). Результаты получены с шагом $\Delta G = 1$ и $\Delta r = 0.001$ при $A_s = 1.5$, $\alpha = 0.1$, $t_0 = 25$, $\Delta t = 6$, $\omega_F = 1$

Ускорение вычислений с использованием библиотеки Numba

```
# подключение библиотеки Numba
import numba as nb

# подключение библиотеки numbalsoda
from numbalsoda import lsoda_sig, lsoda, dop853
from numba import njit, cfunc
```

- Библиотека `Numba` позволяет осуществлять JIT-компиляцию (Just-In-Time compilation, компиляция «точно в нужное время»). JIT-компиляция в процессе выполнения программы транслирует Python и NumPy код в быстрый машинный код.
- Декоратором `@njit` отмечается функция для оптимизации JIT-компилятором Numba.
- Декоратор `@njit(parallel=True)`. Добавление параметра `parallel=True` указывает на подключение автоматического распараллеливания и других оптимизаций для расчетов на CPU.

```
from numba import config, njit, threading_layer, set_num_threads, get_num_threads
# set the threading layer before any parallel target compilation
config.THREADING_LAYER = 'threadsafe'
```

```
@njit
def I_pulse(params):
    ...
    return out
```

```
@njit(parallel=True)
def comput_parallel(params):
    ...
    return out
```

SciPy `solve_ivp()` → numbalsoda `lsoda(funcptr, s0, t_e, rtol = 1e-8, atol = 1e-8, data = data)`

Ускорение с Joblib ~ 35 раз
Ускорение с Numba: `@njit` + `@njit(parallel=True)` ~ 300 раз