Решение дифференциальных уравнений средствами языка программирования.

Фролов А.А., 2к, ИВТ-2

Введение

Решение дифференциальных уравнений — важнейшая часть моделирования процессов в физике, биологии, химии и инженерных науках.
Современные вычислительные средства позволяют не только находить аналитические решения, но и численно моделировать динамику процессов, когда аналитическое решение невозможно.

Для выполнения лабораторной работы выбран Python, поскольку он:

- Имеет понятный синтаксис;
- Поддерживает научные библиотеки `NumPy`, `SciPy`, `Matplotlib`;
- Позволяет строить графики и анализировать данные в интерактивном режиме.

Используемые библиотеки и их назначение

Библиотека	Назначение
NumPy (`numpy`)	Работа с массивами, вычисления логарифмов, экспонент, создание диапазонов значений.
SciPy (`scipy.integrate`)	Решение дифференциальных уравнений (функция `solve_ivp`) и интегралов (`quad`).
Matplotlib (`matplotlib.pyplot`)	Построение графиков, визуализация зависимостей.

Основные команды и их назначение

1. `numpy.linspace(start, stop, num)`

Создаёт массив из `num` равномерно распределённых чисел между `start` и `stop`.
Применяется для задания временного диапазона.

2. `scipy.integrate.solve_ivp(func, t_span, y0, t_eval=None)`

Функция численного решения обыкновенных дифференциальных уравнений (ОДУ).
Реализует методы Рунге-Кутты (по умолчанию 5-го порядка).

Параметры:

- `func(t, y)` — правая часть уравнения (`dy/dt = f(t, y)`);
- `t_span=(t0, tf)` — интервал интегрирования;
- `y0` — начальное значение;
- `t_eval` — массив точек, где нужно вычислить решение.

Возвращает:

- `solution.t` — массив времени;
- `solution.y` — массив значений функции.

3. `scipy.integrate.quad(func, a, b)`

Численное интегрирование функции `func` на интервале `[a, b]`.
Возвращает значение интеграла и оценку ошибки.

4. `matplotlib.pyplot.plot(x, y)`

Построение графика зависимости `y(x)`.

Цель работы

Изучить методы численного решения дифференциальных уравнений, применить их для анализа физических и биологических процессов и визуализировать полученные зависимости.

Задача 1. Остывание кофе

Условие:
Исследовать процесс остывания кофе, находящегося при начальной температуре `T0 = 83 °C` в комнате с температурой `22 °C`.
Коэффициент остывания `r = 0.0373`.
Определить, через сколько минут кофе остынет до `50 °C`.

Теоретическая основа

Процесс описывается законом Ньютона:

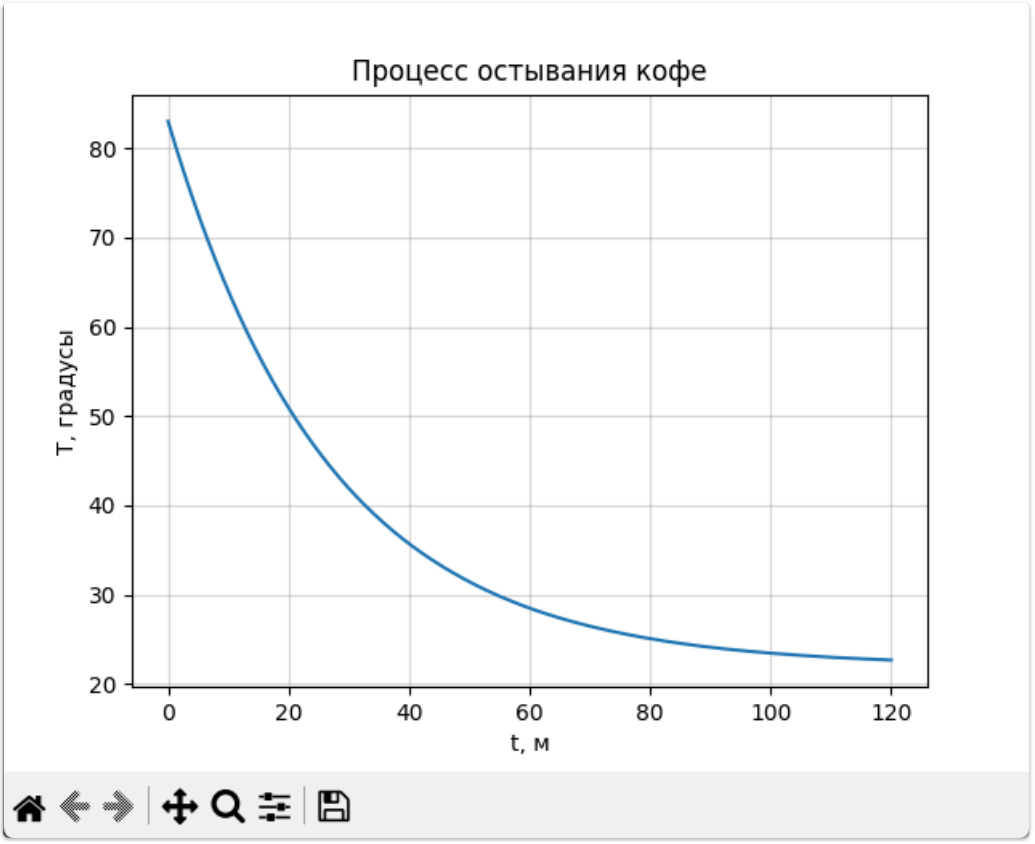
$$\frac{dT}{dt} = -r(T - T_c)$$

Код программы

```python
import numpy as np
from scipy.integrate import solve_ivp
import matplotlib.pyplot as plt

def dTdt(t, T):
    return -r * (T - Ts)

Ts = 22
T0 = 83
r = 0.0373
T_comfort = 50

t_span = (0, 120)
t_points = np.linspace(0, 120, 10**6)

solution = solve_ivp(dTdt, t_span, [T0], t_eval=t_points)

for i in range(len(solution.t)):
    if solution.y[0][i] ⩽ T_comfort:
        print(f"Комфортная температура кофе: {T_comfort} градусов")
        print(f"Кофе остыл до {solution.y[0][i]:.4f} °C за {solution.t[i]:.1f} минут")
        break

plt.title("Процесс остывания кофе")
plt.xlabel("Время, мин")
plt.ylabel("Температура, °C")
plt.grid(True, alpha=0.5)
plt.plot(solution.t, solution.y[0])
plt.show()
```

Результат работы:

```
Комфортная температура кофе: 50 градусов
Кофе остыл до 49.9999 град. за 20.9 минут
```



Задача 2. Падение болида

Условие:

Найти время падения тела с высоты ($R = 60.27 \cdot r_{зем}$),
где ($r_{зем} = 6.377 \times 10^6$ м).
Сила тяжести изменяется по закону ($g(r) = \frac{k}{r^2}$).

Теоретическая формула

$$t = \int_{r_{зем}}^{R} \sqrt{\frac{rR}{2gr_{зем}^2(R-r)}} dr$$

Код программы

```python
from scipy.integrate import quad
import numpy as np

def f(r_):
    return np.sqrt((r_ * r) / (2 * g * r_earth**2 * (r - r_)))

r_earth = 6.377 * 10**6  # м
r = 60.27 * r_earth
g = 9.81
k = g * (r_earth**2)

t, err = quad(f, r_earth, r)

print(f"Время падения тела = {(t/3600):.0f} часов")
```

Результат работы:

```
Время падения тела = 116 часов
```

Задача 3. Распад лекарства

Условие:

После введения лекарства нужно исследовать, как изменяется его количество во времени.
Период полураспада — 4 часа.
Вывести значения через каждые 2 часа.

Уравнение

$$\frac{dN}{dt} = kN, \quad k = \frac{\ln(0.5)}{4}$$

Код программы

```python
import numpy as np
from scipy.integrate import solve_ivp

def dNdt(t, N):
    return k * N

k = np.log(0.5) / 4
t_span = (0, 20)
N0 = [1]
t_points = list(range(0, 21, 2))

solution = solve_ivp(dNdt, t_span, N0, t_eval=t_points)

for i in range(len(solution.t)):
    print(f"t: {solution.t[i]}, N:{solution.y[0][i]:.3f}")
```

Результат работы:

```
t: 0, N:1.000
t: 2, N:0.707
t: 4, N:0.500
t: 6, N:0.354
t: 8, N:0.250
t: 10, N:0.177
t: 12, N:0.125
t: 14, N:0.088
t: 16, N:0.063
t: 18, N:0.044
t: 20, N:0.031
```

Задача 4. Распад радия

Условие:

Скорость распада радия пропорциональна его текущей массе.
Период полураспада = 1590 лет.
Определить процент массы, распавшейся через 200 лет.

Уравнение

$$\frac{dm}{dt} = -km, \quad k = \frac{\ln(2)}{1590}$$

Код программы

```python
import numpy as np
from scipy.integrate import solve_ivp

def dmdt(t, m):
    return -k * m

k = 0.00044
t_span = (0, 200)
m0 = [1]

solution = solve_ivp(dmdt, t_span, m0, t_eval=[0, 200])

m200 = solution.y[0][1]
percent_decayed = (1 - m200) * 100
percent_remaining = m200 * 100

print(f"Начальная масса: 1, Масса через 200 лет: {m200:.4f}")
print(f"Распадется: {percent_decayed:.2f}%, Останется: {percent_remaining:.2f}%")
```

Результат работы:

```
Начальная масса(в долях): 1, Масса через 200 лет: 0.9158
Распадется: 8.42%, Останется: 91.58%
```

Задача 5. Электрическая цепь

Условие:

В цепи: ($E = 300$ В, $R = 150$ Ом, $L = 30$ Гн).
Определить, за какое время ток достигнет 99% от предельного значения.

Уравнение

$$\frac{dI}{dt} = \frac{E - RI}{L}$$

Код программы

```python
import numpy as np
from scipy.integrate import solve_ivp

def dIdt(t, I):
    return (E - R * I) / L

E = 300
R = 150
L = 30
I_lim = E / R
I_target = 0.99 * I_lim

t_span = (0, 2)
t_points = np.linspace(0, 2, 10**6)
I0 = [0]

solution = solve_ivp(dIdt, t_span, I0, t_eval=t_points)

target_time = None
for i in range(len(solution.t)):
    if solution.y[0][i] ⩾ I_target:
        target_time = solution.t[i]
        break
```

```
24
25  print(f"Ток достиг 99% от предельного значения за {target_time:.4f} секунд")
26  print(f"При t = {target_time:.6f} c, I = {solution.y[0][i]:.6f} A")
```

Результат работы:

```
Ток достиг 99% от предельного значения за 0.9237 секунд
При t = 0.923727 c, I = 1.980000 A
```

Задача 6. Сопротивление воздуха в лесу

Условие:
Скорость ветра уменьшается при прохождении через лес по закону:

$$\frac{dv}{ds} = -kv$$

где ($v_0 = 12$ м/с), ($v_1 = 11.8$ м/с) после 1 м пути.
Найти скорость после 150 м.

Код программы

```python
import numpy as np
from scipy.integrate import solve_ivp

def dvds(s, v):
    return -k * v

v0 = 12
v1 = 11.8
s1 = 1.0

k = -np.log(v1 / v0) / s1

s_span = (0, 150)
v_start = [v0]
s_points = [150]

solution = solve_ivp(dvds, s_span, v_start, t_eval=s_points)

print(f"Скорость ветра после прохождения 150м = {solution.y[0][-1]:.4f} м/с")
```

Результат работы:

```
Скорость ветра после прохождения 150м = 0.9656 м/с
```

Заключение

В лабораторной работе исследованы шесть процессов, описываемых дифференциальными уравнениями:

1. Остывание жидкости (закон Ньютона);
2. Падение болида с учётом гравитации;
3. Распад лекарства в организме;
4. Радиоактивный распад радия;
5. Возрастание тока в индуктивной цепи;
6. Снижение скорости ветра при сопротивлении леса.

Для решения использованы функции `solve_ivp` и `quad` библиотеки `SciPy`, обеспечивающие точное численное интегрирование.
Python показал себя удобным инструментом для моделирования и анализа процессов любой физической природы.