ВЫЧИСЛИТЕЛЬНЫЕ МЕТОДЫ · ЛЕКЦИЯ 1

Python для научных вычислений

Основы языка, NumPy и визуализация данных

Лю Шисян

Кафедра теплофизики (Э6)
МГТУ им. Н.Э. Баумана
Москва / 2026

Место этой лекции в курсе

Лекция 1 — Python для научных вычислений

Python рассматривается как основной инструмент реализации численных методов и анализа результатов.

После лекции вы сможете:

- писать простые программы на Python;
- выполнять научные вычисления с помощью NumPy;
- визуализировать результаты с помощью Matplotlib.

План

TOC

1

Введение в Python

Почему Python используется в научных вычислениях?

Простота и скорость разработки

Простой и читаемый синтаксис позволяет быстро реализовывать алгоритмы и проверять новые идеи.

Интерактивность и визуализация

Jupyter позволяет выполнять код пошагово, а Matplotlib — быстро визуализировать результаты вычислений.

Научная экосистема

NumPy, SciPy и другие библиотеки предоставляют готовые инструменты для численных вычислений, линейной алгебры и анализа данных.

Интеграция и современные вычисления

Python взаимодействует с C/C++, Fortran и CUDA и широко используется в HPC, Data Science, Machine Learning и AI.

Python — универсальный инструмент современных научных вычислений.

Среды разработки Python

VS Code

Универсальная среда для программирования.

- встроенный терминал;
- WSL и SSH;
- расширение Python.

Jupyter Notebook

Интерактивная среда в браузере.

- короткие примеры;
- пошаговые вычисления;
- лабораторные работы.

PyCharm

Полноценная IDE для Python.

- удобна для крупных проектов;
- навигация и отладка кода;
- управление окружением.

Spyder

Научная среда, похожая на MATLAB.

- редактор кода;
- интерактивная консоль;
- просмотр переменных.

В курсе: VS Code — для программ, Jupyter Notebook — для интерактивных вычислений.

Управление окружением с помощью Miniconda

Почему Miniconda?

Miniconda предоставляет менеджер пакетов **conda**, который позволяет создавать независимые Python-окружения и устанавливать необходимые научные библиотеки.

1. Создание окружения

```
conda create -n numcalc python=3.12
```

2. Активация окружения

```
conda activate numcalc
```

3. Установка пакетов

```
conda install numpy scipy matplotlib
conda install jupyter notebook
```

4. Запуск Jupyter Notebook

```
jupyter notebook
```

Jupyter работает локально, а интерфейс открывается в браузере.

2

Основы языка Python

Переменные и числовые типы

Основные числовые типы

- `int` — целые числа;
- `float` — вещественные числа;
- `complex` — комплексные числа;
- `type(x)` — определение типа объекта.

Тип определяется автоматически

В Python не нужно заранее объявлять тип переменной: он определяется по присвоенному значению.

```
a = 5
b = 2.0
z = 3 + 4j

# Определение типа
print(type(a))   # int
print(type(b))   # float
print(type(z))   # complex

# Арифметические операции
a + b    # 7.0   сложение
a / 2    # 2.5   обычное деление
a // 2   # 2     целочисленное деление
a % 2    # 1     остаток
a ** 2   # 25    степень
```

Строки и логические значения

Строки: `str`

Строка — это последовательность символов для хранения текста.

```
# Создание строк
name = "Python"

# Обращение к символам
name[0]        # 'P'

# Срез строки
name[0:3]      # 'Pyt'

# Длина строки
len(name)      # 6
```

Логический тип: `bool`

Логическое выражение принимает одно из двух значений: `True` или `False`.

```
x = 2

# Операции сравнения
x > 0          # True
x == 2         # True
x != 3         # True

# Логические операции
(x > 0) and (x < 5)   # True
(x < 0) or (x == 2)   # True
not (x == 2)          # False
```

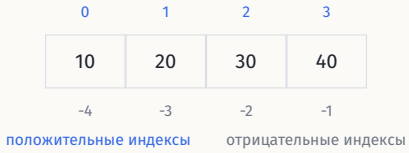
Список `list`: базовая структура данных

Что такое список?

Список — это упорядоченная и изменяемая коллекция объектов.

- элементы имеют определённый порядок;
- индекс начинается с 0;
- поддерживаются срезы;
- элементы списка можно изменять и добавлять.

Индексация списка




```
a = [10, 20, 30, 40]

# Индексация
a[0]       # 10
a[-1]      # 40

# Срез
a[1:3]     # [20, 30]

# Добавление элемента
a.append(50)

# Основные функции
len(a)     # 5
sum(a)     # 150
max(a)     # 50
```

Другие структуры данных

Кортеж `tuple`

Упорядоченная, но неизменяемая коллекция.

```
p = (2.0, 3.0)

# Доступ по индексу
p[0]       # 2.0
p[1]       # 3.0
```

Подходит для данных, которые не должны изменяться.

Словарь `dict`

Хранит данные в виде пар `ключ: значение`.

```
mat = {
    "name": "Si",
    "k": 148
    }

# Доступ по ключу
mat["k"]       # 148
```

Удобен для хранения именованных параметров.

Множество `set`

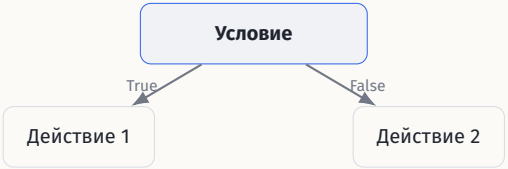
Коллекция уникальных элементов без повторений.

```
s = {1, 2, 2, 3}

print(s)
# {1, 2, 3}

2 in s      # True
```

Удобно для удаления повторяющихся значений.

Условный оператор `if`

Условный оператор позволяет выполнять разные действия в зависимости от значения логического выражения.



В Python блоки программы определяются **отступами**. Обычно используется четыре пробела.

```
T = 420

# Первое условие
if T > 400:
    print(”High temperature”)

# Дополнительное условие
elif T > 300:
    print(”Moderate temperature”)

# Если условия выше не выполнены
else:
    print(”Low temperature”)
```

Порядок проверки:

$$\texttt{if} \rightarrow \texttt{elif} \rightarrow \texttt{else}$$

Цикл `for`

Цикл `for` последовательно перебирает элементы коллекции или заданную последовательность значений.

Перебор элементов

```
data = [1.0, 1.5, 2.0]
for x in data:
    print(x ** 2)
```

Индекс и значение

```
data = [1.0, 1.5, 2.0]
for i, x in enumerate(data):
    print(i, x)
```

Последовательность с `range`

```
for i in range(5):
    print(i)
```

Накопление результата

```
s = 0
for i in range(1, 101):
    s += i
print(s)
```

`for` = повторить действие для каждого элемента

Цикл while, break и continue

Цикл while

Команды выполняются до тех пор, пока условие остаётся истинным.

```
x = 1.0

while x < 10:
    x *= 1.5
    print(x)
```

Управление циклом

continue пропускает текущую итерацию, а break полностью завершает цикл.

```
for i in range(10):
    if i == 3:
        continue   # пропустить 3
    if i == 7:
        break      # завершить цикл
    print(i)
```

Когда использовать?

- число шагов заранее неизвестно;
- остановка определяется условием.

for — перебор последовательности **while — повторение по условию**

Пользовательские функции

Что такое функция?

Функция — это именованный блок кода, который выполняет определённую задачу и может быть вызван многократно.

Функции позволяют:

- повторно использовать код;
- разделять программу на отдельные задачи;
- передавать входные параметры;
- возвращать результат с помощью `return`.

```
# Определение функции
def heat_flux(k, dT, L):
    # Закон Фурье
    q = -k * dT / L
    # Возврат результата
    return q

# Вызов функции
q = heat_flux(150, 20, 0.01)

print(q)   # -300000.0
```

`def` — определить функцию **`return` — вернуть результат**

3

Научные вычисления с NumPy

Почему обычного списка недостаточно?

Python `list`

Универсальная структура данных, но арифметические операции над элементами обычно требуют явного цикла.

```
a = [1, 2, 3, 4]
a + [5, 6] # Объединение списков

b = []
for x in a:
    b.append(x + 1)
```

NumPy `array`

Массив предназначен для численных вычислений и поддерживает векторные операции.

```
import numpy as np
a = np.array([1, 2, 3, 4])

# Поэлементные операции
a + 1  # [2 3 4 5]
a * 2  # [2 4 6 8]
```

NumPy позволяет выполнять операции сразу над всем массивом

Создание массивов NumPy

Базовые способы создания

NumPy предоставляет готовые функции для быстрого создания массивов.

```
import numpy as np

# Из списка
a = np.array([1, 2, 3, 4])

# Массив из нулей
b = np.zeros(5)

# Массив из единиц
c = np.ones(5)
```

Создание численной сетки

Для координат, времени и параметров особенно часто используются `arange` и `linspace`.

`arange`: задаём шаг

```
x = np.arange(0, 10, 2)

# [0 2 4 6 8]
```

`linspace`: задаём число точек

```
x = np.linspace(0, 1, 6)

# [0.  0.2 0.4 0.6 0.8 1. ]
```

Свойства массива

Основные атрибуты

Каждый массив NumPy имеет информацию о своей размерности, форме и типе данных.

- `shape` — размеры массива по осям;
- `ndim` — число измерений;
- `size` — общее число элементов;
- `dtype` — тип данных элементов.

$$\texttt{shape} = (2, 3)$$

2 строки × 3 столбца

Форма массива определяет структуру данных и совместимость операций.

```
a = np.array([
    [1, 2, 3],
    [4, 5, 6]
])

print(a.shape)
# (2, 3)

print(a.ndim)
# 2

print(a.size)
# 6

print(a.dtype)
# int64
```

Индексация и срезы NumPy

Одномерный массив

Индексация позволяет выбрать отдельный элемент, а срез — часть массива.

```
a = np.array([10, 20, 30, 40, 50])

# Отдельные элементы
a[0]       # 10
a[-1]      # 50

# Срезы
a[1:4]     # [20 30 40]
a[::2]     # [10 30 50]
```

`a[start:stop:step]`

Двумерный массив

Для двумерного массива используется запись

`A[строка, столбец].`

```
A = np.array([
    [1, 2, 3],
    [4, 5, 6]
])

A[0, 1]    # 2
A[1, :]    # [4 5 6]
A[:, 1:]
# [[2 3]
#  [5 6]]
```

Логические маски

Логическая маска — это массив значений `True` и `False`, который используется для выбора элементов по условию.


```
import numpy as np

T = np.array([280, 310, 450, 295])

mask = T > 300
print(mask)
# [False  True  True False]

print(T[mask])
# [310 450]
```

Краткая запись:

```
T[T > 300]
```

Сначала создаётся маска, затем по ней выбираются элементы массива.

Векторизация и поэлементные операции

NumPy позволяет выполнять одну и ту же математическую операцию сразу над всеми элементами массива без явного цикла Python.

Явный цикл Python

```
x = np.linspace(0, 1, 1000)
y = np.zeros_like(x)

# Обрабатываем каждый элемент
for i in range(len(x)):
    y[i] = np.sin(x[i]) ** 2
```

Векторная запись NumPy

```
x = np.linspace(0, 1, 1000)

# Операция над всем массивом
y = np.sin(x) ** 2
```

$$y_i = \sin^2(x_i), \qquad i = 0, \dots, N-1$$

$$\mathbf{y} = \sin^2(\mathbf{x})$$

Обе записи дают одинаковый массив y, но векторная форма проще и обычно быстрее.

Широковещание массивов (broadcasting)

Широковещание позволяет выполнять операции над массивами разной формы, если их размеры совместимы.

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}10 & 20 & 30\end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix} + \begin{bmatrix}10 & 20 & 30\\10 & 20 & 30\end{bmatrix} = \begin{bmatrix}11 & 22 & 33\\14 & 25 & 36\end{bmatrix}$$

Массив b распространяется по строкам.

```
A = np.array([
    [1, 2, 3],
    [4, 5, 6]
])
b = np.array([10, 20, 30])

C = A + b
print(C)
```

$$\texttt{A.shape} = (2, 3), \quad \texttt{b.shape} = (3,)$$

Размеры совместимы, поэтому broadcasting возможен.

Широковещание упрощает операции над массивами разных форм.

Изменение формы массива

Изменение формы: `reshape`

Функция `reshape` изменяет форму массива, не изменяя общее число его элементов.

```
a = np.arange(6)
A = a.reshape(2, 3)

print(A)
# [[0 1 2]
#  [3 4 5]]
```

$$(6,) \quad \longrightarrow \quad (2, 3)$$

Транспонирование

Транспонирование меняет местами строки и столбцы.

```
AT = A.T

print(AT)
# [[0 3]
#  [1 4]
#  [2 5]]
```

$$(2, 3) \quad \longrightarrow \quad (3, 2)$$

Объединение массивов

NumPy позволяет объединять несколько массивов вдоль разных направлений.

`concatenate`: в один массив

```
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
z = np.concatenate((x, y))
print(z)
# [1 2 3 4 5 6]
```

`column_stack`: по столбцам

```
B = np.column_stack((x, y))
print(B)
# [[1 4]
#  [2 5]
#  [3 6]]
```

`vstack`: по строкам

```
A = np.vstack((x, y))
print(A)
# [[1 2 3]
#  [4 5 6]]
```

Разница

- `concatenate` — объединение;
- `vstack` — строки;
- `column_stack` — столбцы.

Статистические функции NumPy

- `np.mean(a)` — среднее;
- `np.std(a)` — стандартное отклонение;
- `np.sum(a)` — сумма;
- `np.min(a)`, `np.max(a)` — экстремумы;
- `np.argmin(a)`, `np.argmax(a)` — их индексы.

```
T = np.array([300, 305, 302, 310])

# Среднее и медиана
np.mean(T)       # 304.25
np.median(T)     # 303.5

# Разброс данных
np.std(T)

# Минимум и максимум
np.min(T)        # 300
np.max(T)        # 310

# Индекс максимума
np.argmax(T)     # 3
```

Скалярное произведение векторов

Для двух векторов одинаковой длины $\mathbf{a} = (a_1, a_2, \dots, a_n), \mathbf{b} = (b_1, b_2, \dots, b_n)$, скалярное произведение определяется как

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$$

В NumPy

```
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])

# Скалярное произведение
a @ b          # 32
np.dot(a, b)   # 32
```

Пример вычисления

$$\begin{aligned} \mathbf{a} \cdot \mathbf{b} &= 1 \cdot 4 + 2 \cdot 5 + 3 \cdot 6 \\ &= 4 + 10 + 18 \\ &= 32 \end{aligned}$$

Геометрический смысл

$$\mathbf{a} \cdot \mathbf{b} = |\mathbf{a}| \, |\mathbf{b}| \cos\theta$$

Матричное произведение

$$\overset{2\times 3}{A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}} \quad \times \quad \overset{3\times 2}{B=\begin{bmatrix}1 & 2\\ 3 & 4\\ 5 & 6\end{bmatrix}} \quad = \quad \overset{2\times 2}{C=\begin{bmatrix}22 & 28\\ 49 & 64\end{bmatrix}}$$

Правило размерностей

$$(m\times n)(n\times p)\longrightarrow(m\times p)$$

Внутренние размерности должны совпадать:

$$3=3$$

а внешние определяют форму результата:

$$2\times 2.$$

```
A = np.array([[1, 2, 3],[4, 5, 6]])

B = np.array([[1, 2],[3, 4],[5, 6]])

C = A @ B

print(C)
# [[22 28]
#  [49 64]]
```

NumPy и MATLAB: важные различия

Операция	NumPy	MATLAB
Матричное умножение	`A @ B`	`A * B`
Поэлементное умножение	`A * B`	`A .* B`
Скалярное произведение	`u @ v`	`dot(u,v)`
Транспонирование	`A.T`	`A.'`

Главное различие

В NumPy

`*` — **поэлементное умножение**,

`@` — **матричное умножение**.

Одномерный массив

В NumPy

$$\texttt{u.shape} = (N,)$$

не является ни строкой $(1, N)$, ни столбцом $(N, 1)$.

Генерация случайных чисел в NumPy

```
# Создаем генератор
rng = np.random.default_rng(seed=42)

# Равномерное распределение
u = rng.uniform(0, 1, 1000)

# Нормальное распределение
n = rng.normal(0, 1, 1000)

# Случайные целые числа
i = rng.integers(1, 7, size=20)

# Случайный выбор
c = rng.choice(["A", "B", "C"], size=10)
```

- `uniform` — равномерное распределение;
- `normal` — нормальное распределение;
- `integers` — случайные целые числа;
- `choice` — случайный выбор.

Зачем нужен `seed`?

Одинаковое значение `seed` позволяет воспроизводить одну и ту же последовательность случайных чисел.

Случайные числа будут основой методов Монте-Карло в лекциях 5–6.

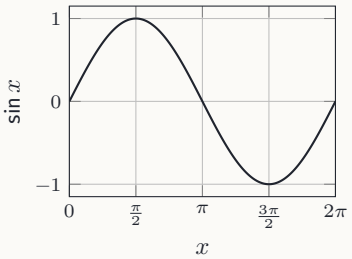
4

Визуализация с Matplotlib

Первый график в Matplotlib

Matplotlib строит график по массивам координат x и y.

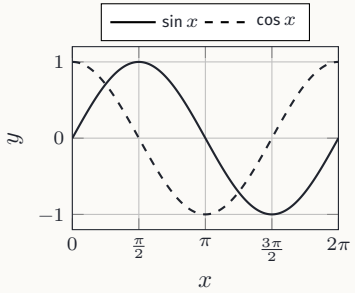
```
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(0, 2*np.pi, 200)
y = np.sin(x)

plt.plot(x, y)        # Строим график
plt.xlabel("x")       # Ось x
plt.ylabel("sin(x)")  # Ось y
plt.grid()            # Сетка
plt.show()            # Показываем график
```



Несколько кривых и оформление графика

Каждый вызов `plot()` добавляет новую кривую на текущий график.

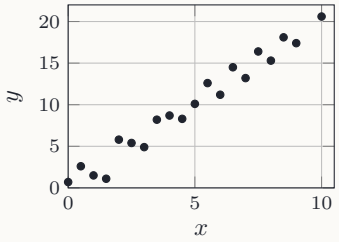
```
x = np.linspace(0, 2*np.pi, 200)

plt.figure(figsize=(6, 4))

plt.plot(x, np.sin(x), label="sin(x)")
plt.plot(x, np.cos(x), "--", label="cos(x)")

plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.grid()
plt.tight_layout()
plt.savefig("trig.png", dpi=300)
plt.show()
```



Точечный график

Точечный график показывает отдельные наблюдения и позволяет увидеть зависимость между двумя величинами.

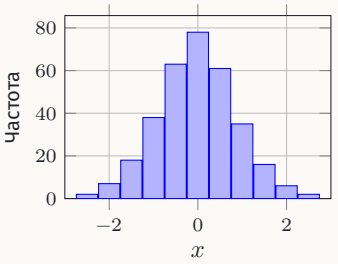
```
rng = np.random.default_rng(1)

x = np.linspace(0, 10, 50)
y = 2*x + rng.normal(0, 2, 50)

plt.scatter(x, y)

plt.xlabel("x")
plt.ylabel("y")
plt.grid()
plt.show()
```



Гистограмма

Гистограмма показывает, как значения данных распределены по заданным интервалам.

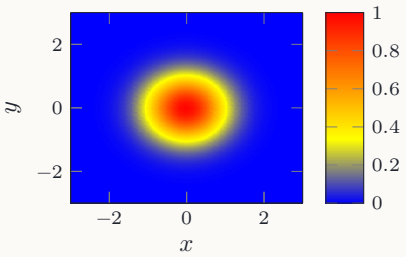
```
rng = np.random.default_rng(1)

data = rng.normal(0, 1, 1000)

plt.hist(data, bins=30)

plt.xlabel(”x”)
plt.ylabelЧастота(””)
plt.grid()
plt.show()
```



Визуализация двумерного поля

Функция `contourf` позволяет визуализировать скалярную величину, заданную на двумерной сетке.

```
x = np.linspace(-5, 5, 100)
y = np.linspace(-5, 5, 100)

X, Y = np.meshgrid(x, y)

Z = np.exp(-(X**2 + Y**2))

plt.contourf(X, Y, Z, levels=30)

plt.colorbar(label="T")
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



Цвет показывает значение величины $Z(x, y)$.

Сохранение и загрузка численных данных

Текстовые данные: `txt`

Удобны для просмотра и обмена данными с другими программами.

```
x = np.linspace(0, 1, 11)
y = x ** 2

# Объединяем два столбца
data = np.column_stack((x, y))

# Сохраняем
np.savetxt("data.txt", data)

# Загружаем
data = np.loadtxt("data.txt")
```

Формат NumPy: `npy`

Компактный формат для сохранения массивов NumPy с их формой и типом данных.

```
rng = np.random.default_rng(1)

A = rng.random((100, 100))

# Сохраняем массив
np.save("A.npy", A)

# Загружаем массив
B = np.load("A.npy")
```

5

Практика и итоги

Пример: охлаждение тела

Закон Ньютона для охлаждения

Скорость изменения температуры пропорциональна разности между температурой тела и окружающей среды:

$$\frac{dT}{dt} = -k\,(T - T_\infty)\,.$$

Аналитическое решение:

$$\boxed{T(t) = T_\infty + (T_0 - T_\infty)\,e^{-kt}}$$

Начальная температура

$T_0 = 500$ K

Окружающая среда

$T_\infty = 300$ K

Коэффициент охлаждения

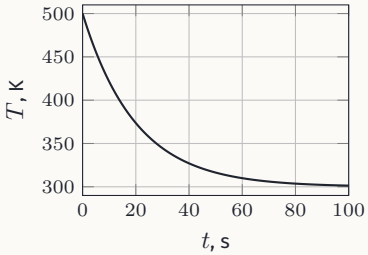
$k = 0.05$ s^{-1}

Сейчас вычислим аналитическое решение с помощью NumPy и визуализируем его в Matplotlib.

Пример: реализация на Python

NumPy используется для вычисления температуры во множество моментов времени.
Matplotlib — для визуализации.

```
import numpy as np
import matplotlib.pyplot as plt

T0 = 500.0
T_inf = 300.0
k = 0.05
t = np.linspace(0, 100, 300)
T = T_inf + (T0 - T_inf) * np.exp(-k*t)

plt.plot(t, T)
plt.xlabel("t, s")
plt.ylabel("T, K")
plt.grid()
plt.show()
```



$T(t) \to T_\infty$ **при увеличении времени.**

Что уже умеем после этой лекции?

Мы научились превращать известную математическую модель в вычисления, графики и численные данные с помощью Python.

А что делать, если аналитического решения нет?

Тогда необходимы численные методы.

Задания для самостоятельной работы

1. Графики функций

Постройте на одном рисунке

$$y_1 = \sin x, \qquad y_2 = e^{-0.2x} \sin x, \qquad x \in [0, 20].$$

Добавьте подписи осей, легенду и сетку.

2. Двумерное поле

Постройте карту поля

$$T(x, y) = 300 + 100 \exp\left[-\frac{x^2 + y^2}{2}\right], \qquad x, y \in [-5, 5].$$

Итоги лекции

Python

- переменные и типы данных;
- условия и циклы;
- пользовательские функции.

Matplotlib

- линейные и точечные графики;
- гистограммы;
- двумерные поля.

NumPy

- массивы и индексация;
- векторизация;
- операции с векторами и матрицами.

Рабочий процесс

модель $\longrightarrow$ расчёт $\longrightarrow$ график $\longrightarrow$ данные

Следующая лекция: Linux, WSL и вычислительная среда