

Лекция 1

Python для научных вычислений

Лю Шисян

Кафедра теплофизики (Э6)

План лекции

1.1 Установка в Windows	Создать окружение Python и запустить блокнот Jupyter.
1.2 Основы Python	Записать алгоритм с помощью условий, циклов и функций.
1.3 NumPy	Выполнить поэлементные операции с массивами и перемножить матрицы.
1.4 Matplotlib	Построить график, показать погрешность измерений и сохранить рисунок.

Приложение Умножение в NumPy и MATLAB Команды Linux

Репозиторий кода `https://github.com/lyushisyan/computational-methods-course`

1.1 Установка Python в Windows

Установим Miniconda и создадим окружение с библиотеками для расчётов. Запустим блокнот Jupyter и проверим, что в нём выполняется код на Python.

Скачайте Miniconda для Windows

Miniconda включает Python и менеджер окружений `conda`. Для установки понадобится один файл `.exe`.

1. Узнайте тип процессора

Нажмите `Win+R`, введите `ms-settings:about` и нажмите `Enter`. В открывшемся окне найдите строку «Тип системы».

2. Выберите установщик

Откройте список файлов и скачайте вариант для своего процессора:

`https://repo.anaconda.com/miniconda/`

Тип процессора	**Имя файла**
x64 (Intel или AMD)	`Miniconda3-latest-Windows-x86_64.exe`
ARM64	`Miniconda3-latest-Windows-arm64.exe`

Установите и проверьте Miniconda

1. Установите Miniconda

Дважды щёлкните по скачанному файлу `.exe`. Выберите **Just Me**, чтобы установить Miniconda для своей учётной записи. Оставьте папку и параметры по умолчанию, нажмите **Install**, затем **Finish**.

2. Откройте Anaconda Prompt

В меню «Пуск» найдите **Anaconda Prompt** и откройте его. Это командная строка Windows, в которой уже доступна команда `conda`.

3. Проверьте установку

Введите команду и нажмите `Enter`.

```
conda --version
```

Если в ответ появился номер версии `conda`, можно создавать окружение.

Создайте окружение с Python 3.12

1. Создайте окружение

В Anaconda Prompt выполните команду. При запросе `Proceed ([y]/n)?` подтвердите выбор клавишей `Enter`.

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

2. Активируйте окружение

После этой команды в начале строки появится `(numcalc)`.

```
conda activate numcalc
```

3. Проверьте версию Python

Ответ должен начинаться с `Python 3.12`.

```
python --version
```

При следующем запуске Anaconda Prompt снова выполните `conda activate numcalc`.

Запустите первый блокнот Jupyter

1. Установите пакеты в Anaconda Prompt

Убедитесь, что в начале строки указано `(numcalc)`, и выполните команду:

```
conda install numpy matplotlib notebook
```

2. Запустите Jupyter Notebook

После установки введите следующую команду. Оставьте окно Anaconda Prompt открытым.

```
jupyter notebook
```

3. Выполните первую ячейку в браузере

Создайте блокнот с ядром Python 3 и введите в первую ячейку две строки:

```
import numpy as np
print(np.arange(3))
```

Нажмите `Shift+Enter`. Под ячейкой появится `[0 1 2]`.

1.2 Основы языка Python

Запишем алгоритм на Python с помощью переменных, условий и циклов. Оформим вычисления в виде функций, чтобы применять их к разным данным.

Переменные и числовые типы

Значения и типы

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

print(type(a))
print(type(b))
print(type(z))
```

```
<class 'int'>
<class 'float'>
<class 'complex'>
```

`j` обозначает мнимую единицу.

Арифметические операции

При `a = 5`, `b = 2.0`:

Выражение	Результат	Операция
`a + b`	`7.0`	Сложение
`a - b`	`3.0`	Вычитание
`a * b`	`10.0`	Умножение
`a / b`	`2.5`	Деление
`a // b`	`2.0`	Деление с округлением вниз
`a % b`	`1.0`	Остаток от деления
`a ** b`	`25.0`	Возведение в степень

Строки

`str`

```
s1 = "hello "
s2 = 'world'
s = s1 + s2
```

В конце `s1` есть пробел: он тоже считается символом.

Выражение	Результат в Notebook	Операция
`s`	`'hello world'`	Соединение строк
`s1 * 3`	`'hello hello hello '`	Повторение
`s1[0]`	`'h'`	Первый символ
`s1[0:3]`	`'hel'`	Символы с индексами 0, 1, 2
`len(s1)`	`6`	Длина с учётом пробела
`s.split()`	`['hello', 'world']`	Разделение на слова
`s.replace('h', 'H')`	`'Hello world'`	Замена символа

Индексы начинаются с 0. Правая граница среза не включается.

Булевы значения

bool

Логические операции

```
x = True
y = False
```

`x and y`	`False`	И
`x or y`	`True`	ИЛИ
`not x`	`False`	НЕ

Преобразование в bool

`a = 10`	`bool(a)`	`True`
`b = 0`	`bool(b)`	`False`
`c = ''`	`bool(c)`	`False`
`d = []`	`bool(d)`	`False`
`e = None`	`bool(e)`	`False`

Сравнение чисел

```
a = 2
b = 3
```

`a == b`	`False`	`a != b`	`True`
`a > b`	`False`	`a < b`	`True`

= —присваивание; == —сравнение.

Условие в программе

```
age = 20
if age >= 18:
    print("Взрослый")
else:
    print("Несовершеннолетний")
```

Вывод: `Взрослый`

Список и кортеж

Список `list`

```
a = [1, 2, 3, 4]
print(a[0])     # 1
print(a[1:3])   # [2, 3]
```

В списке можно заменить элемент или добавить новый в конец.

```
a[0] = 10
a.append(5)
print(a)  # [10, 2, 3, 4, 5]
```

Кортеж `tuple`

```
b = (1, 2, 3, 4)
print(b[0])     # 1
print(b[1:3])   # (2, 3)
```

Попытка заменить элемент кортежа вызовет ошибку.

```
b[0] = 10   # TypeError
```

В обеих последовательностях индексы начинаются с нуля. Срез `[1:3]` берёт элементы с индексами 1 и 2.

Словарь

dict

Словарь хранит пары **ключ–значение**. Ключи уникальны.

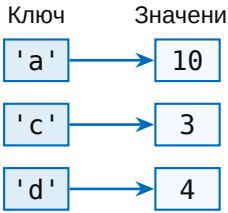
```
dic = {'a': 1, 'b': 2, 'c': 3}
```

`dic['b']` ⟶ 2.

```
dic['d'] = 4
dic['a'] = 10
del dic['b']
```

Добавляем `'d'`, меняем значение `'a'`, удаляем `'b'`.

После изменений



Метод	Что возвращает
`dic.keys()`	Ключи
`dic.values()`	Значения
`dic.items()`	Пары ключ–значение

Пользовательская функция: закон Фурье

Теплопроводность плоского слоя

При линейном профиле температуры:

$$q = -k\frac{\Delta T}{L}, \qquad \Delta T = T_2 - T_1.$$

- k —теплопроводность, Вт/(м · К);
- ΔT —разность температур, К;
- L —толщина слоя, м.

q —плотность теплового потока, Вт/м^2.

Знак минус: тепло переносится в сторону меньшей температуры.

Определение и вызов функции

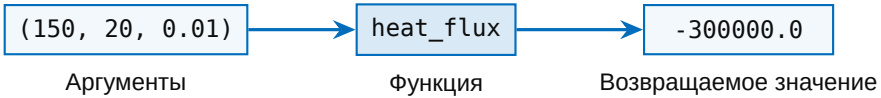
```
def heat_flux(k, dT, L):
    q = -k * dT / L
    return q

q = heat_flux(150, 20, 0.01)
print(q)
```

Вывод Python

```
-300000.0
```

`def` определяет функцию.
`return q` возвращает вычисленное значение.



Анонимные функции

lambda создаёт функцию из одного выражения.

Квадрат одного числа

```
square = lambda x: x ** 2
print(square(5))
```

Вывод: 25

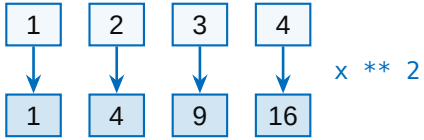


x —параметр.
Значение x ** 2 становится результатом функции.

Квадрат каждого элемента

```
nums = [1, 2, 3, 4]
squared = list(map(
    lambda x: x ** 2, nums))
print(squared)
```

Вывод: [1, 4, 9, 16]



map применяет функцию к каждому элементу.
list собирает результаты в список.

Цикл `for`: перебор значений

Элементы списка

```
temperatures = [280, 300, 320]
for T in temperatures:
    print(T)
```

```
280
300
320
```

Ключи и значения словаря

```
T = {'A': 280, 'B': 300}
for point, value in T.items():
    print(point, value)
```

```
A 280
B 300
```

В цикле `for i in range(3):` переменная `i` принимает значения 0, 1 и 2.

Циклы: `while`, `break`, `continue`

Условие `while`

Цикл выполняется, пока условие истинно.

```
i = 0
while i < 5:
    print(i)
    i += 1
```

Выход по `break`

При `i == 2` цикл завершается.

```
for i in range(5):
    if i == 2:
        break
    print(i)
```

Пропуск по `continue`

При `i == 2` цикл переходит к следующему числу.

```
for i in range(5):
    if i == 2:
        continue
    print(i)
```

Какие числа напечатает каждый цикл?

`0, 1, 2, 3, 4` `0, 1` `0, 1, 3, 4`

Каждый вызов `print(i)` выводит число на отдельной строке.

1.3 Научные вычисления с NumPy

Организуем численные данные в массивы. Научимся выбирать нужные элементы и выполнять вычисления над целыми массивами.

Сложение списков и массивов NumPy

```
import numpy as np
a = [1, 2, 3, 4]
b = [2, 3, 4, 5]
```

Списки соединяются

```
print(a + b)
```

```
[1, 2, 3, 4, 2, 3, 4, 5]
```

Сначала идут все элементы a, затем все элементы b.

Массивы складываются поэлементно

```
x = np.array(a)
y = np.array(b)
print(x + y)
print(x + 1)
```

```
[3 5 7 9]
[2 3 4 5]
```

Тип элементов массива

`dtype` задаёт тип всех элементов. Метод `fill` заполняет массив одним значением и сохраняет его тип.

Массив целых чисел

```
a = np.zeros(4, dtype=int)
a.fill(2.5)
print(a)
```

```
[2 2 2 2]
```

При записи в целочисленный массив дробная часть отбрасывается.

Переход к вещественным числам

```
b = a.astype(float)
b.fill(2.5)
print(b)
```

```
[2.5 2.5 2.5 2.5]
```

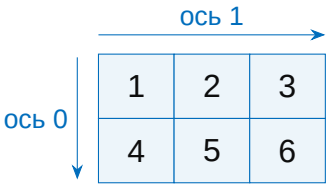
`astype(float)` создаёт массив, в котором можно хранить дробные значения.

`np.zeros(4)` и `np.ones(4)` по умолчанию создают массивы типа `float64`.

Форма массива и его оси

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

```
(2, 3)
2
6
```

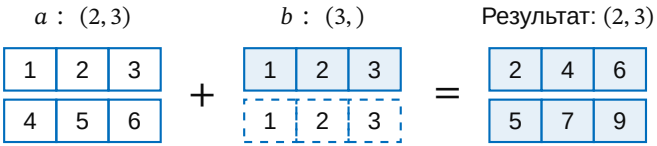


В кортеже `(2, 3)` первое число даёт количество строк, второе —количество столбцов.

`ndim` —число осей, `size` —число элементов. Здесь $2 \times 3 = 6$.

Broadcasting: прибавляем строку к матрице

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



Вектор b прибавляется к **каждой строке** массива a.

Размеры сравниваются справа: они должны совпадать, либо один из них равен 1.

Равномерная сетка: шаг и число точек

Известен шаг

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

```
[0 2 4 6 8]
```

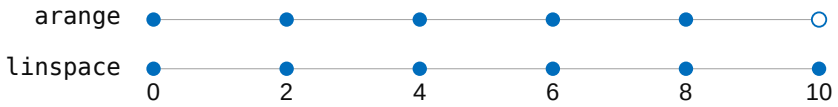
Отсчёт идёт от 0 с шагом 2. Правая граница 10 исключена.

Известно число точек

```
x = np.linspace(0, 10, 6)
print(x)
```

```
[ 0.  2.  4.  6.  8. 10.]
```

Шесть точек образуют пять интервалов. Обе границы включены по умолчанию.



Массивы из списковых выражений

Линейная последовательность

```
a = np.array([
    1 + i for i in range(10)
])
print(a)
```

```
[ 1  2  3  4  5  6  7  8  9 10]
```

Квадраты чисел

```
a = np.array([
    (1 + i)**2 for i in range(10)
])
```

```
1, 4, 9, 16, 25, 36,
49, 64, 81, 100
```

Двумерный массив: $a_{ij} = ij$

```
a = np.array([
    [i * j for j in range(4)]
    for i in range(3)
])
```

$i \backslash j$	0	1	2	3
$i = 0$	0	0	0	0
$i = 1$	0	1	2	3
$i = 2$	0	2	4	6

Внутренний цикл создаёт строку, внешний — три строки.

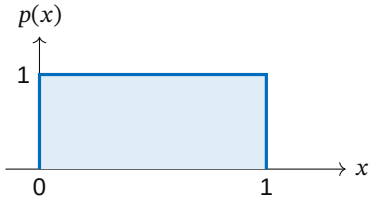
Случайные числа: два распределения

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

Равномерное распределение

```
a = rng.random(10)
```

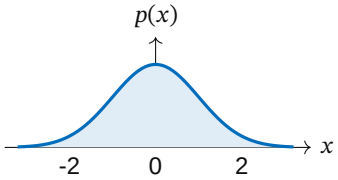
10 чисел из интервала $[0, 1)$.



Нормальное распределение

```
b = rng.normal(0, 1, size=10)
```

Среднее $\mu = 0$, стандартное отклонение $\sigma = 1$.



При повторном запуске всех трёх команд с тем же начальным значением 1 генератор воспроизводит обе выборки.

Случайные целые числа и выбор элементов

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

Целые числа из интервала

```
a = rng.integers(1, 20, size=6)
print(a)
```

```
[ 9 10 15 19  1  3]
```

`size=6` задаёт число элементов. Каждое значение лежит между 1 и 19 включительно.

Выбор из заданного набора

```
values = [10, 20, 30, 40]
b = rng.choice(values, size=3,
               replace=False)
print(b)
```

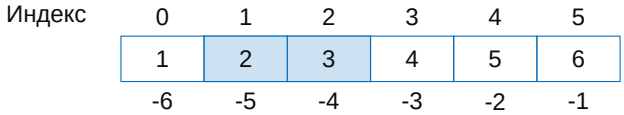
```
[10 30 20]
```

`replace=False` задаёт выбор без возвращения.

Чтобы получить показанные результаты, выполните команды по порядку: сначала `integers`, затем `choice`.

Индексация и срезы: одномерный массив

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




```
print(a[1:3])   # [2 3]
print(a[1:-2])  # [2 3 4]
print(a[::2])   # [1 3 5]
```

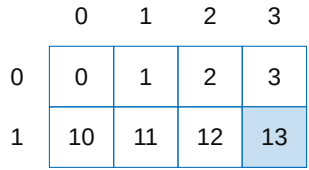
```
a[0] = 10
print(a)
# [10  2  3  4  5  6]
```

`a[start:stop:step]`: индекс `stop` не входит в срез.

Индексация: строки и столбцы

```
a = np.array([[0, 1, 2, 3],
              [10, 11, 12, 13]])
print(a[1, 3])
print(a[1])
print(a[:, 1])
```

```
13
[10 11 12 13]
[ 1 11]
```

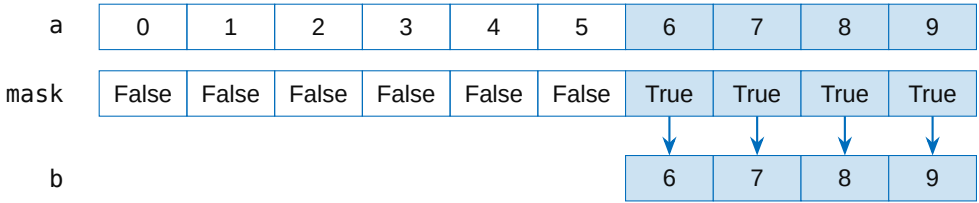


`:` выбирает все позиции по оси.

```
a[1, 3] = -1
# [[ 0  1  2  3]
#  [10 11 12 -1]]
```

Логическая маска: выбор по условию

```
a = np.arange(0, 10)
mask = (a > 5)
b = a[mask]
print(b)  # [6 7 8 9]
```



В результат попадают только элементы, которым соответствует `True`.

Срез массива и независимая копия

Обычный срез разделяет данные

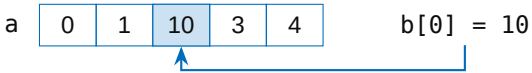
```
a = np.array([0, 1, 2, 3, 4])
b = a[2:4]
b[0] = 10
print(a)
```

```
[ 0  1 10  3  4]
```

`copy()` создаёт свои данные

```
a = np.array([0, 1, 2, 3, 4])
b = a[2:4].copy()
b[0] = 10
print(a)
```

```
[0 1 2 3 4]
```



Для списка `a = [1, 2, 3, 4, 5]` срез `b = a[2:4]` создаёт новый список: присваивание `b[0] = 10` не меняет `a`.

Объединение массивов и выбор оси

```
x = np.array([[0, 1, 2], [10, 11, 12]])
y = np.array([[50, 51, 52], [60, 61, 62]])
```

Каждый массив содержит две строки и три столбца.

Вызов	Форма	Расположение данных
`np.concatenate((x, y), axis=0)`	`(4, 3)`	Строки y под строками x
`np.concatenate((x, y), axis=1)`	`(2, 6)`	Столбцы y справа от x
`np.stack((x, y), axis=0)`	`(2, 2, 3)`	Два слоя по две строки и три столбца

`concatenate` соединяет массивы вдоль существующей оси. `stack` добавляет новую ось и требует одинаковой формы входных массивов.

Изменение формы и транспонирование

`reshape` возвращает массив новой формы

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

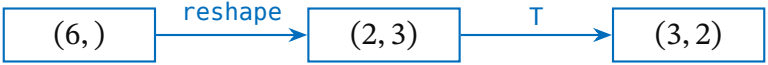
```
[0 1 2 3 4 5]
[[0 1 2]
 [3 4 5]]
```

T: строки становятся столбцами

```
print(b.T)
```

```
[[0 3]
 [1 4]
 [2 5]]
```

Присваивание `a.shape = (2, 3)` меняет форму самого `a`.



Сортировка: новый массив или изменение на месте

```
data = np.array([3, 1, 4, 1, 5])
```

np.sort(data)

```
sorted_data = np.sort(data)
print(sorted_data)
print(data)
```

```
[1 1 3 4 5]
[3 1 4 1 5]
```

Исходный массив сохраняется.

data.sort()

```
data.sort()
print(data)
```

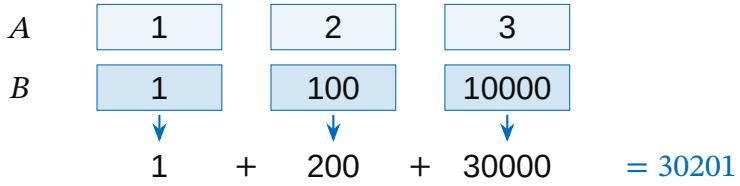
```
[1 1 3 4 5]
```

Меняется сам массив `data`.



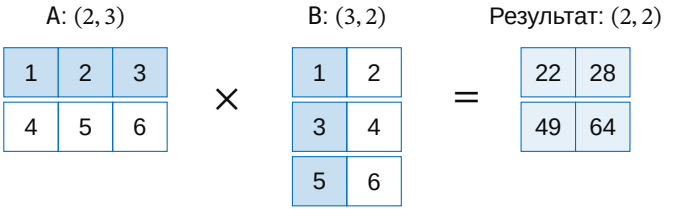
Скалярное произведение векторов

```
A = np.array([1, 2, 3])
B = np.array([1, 100, 10000])
print(np.dot(A, B))  # 30201
```



$A \cdot B = \sum_i A_i B_i$ — результатом является одно число.

Матричное произведение



$$C_{00} = 1 \cdot 1 + 2 \cdot 3 + 3 \cdot 5 = 22$$

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

`A @ B` умножает матрицы. Оператор * умножает соответствующие элементы. Для форм $(2, 3)$ и $(3, 2)$ такое умножение невозможно.

1.4 Визуализация с Matplotlib

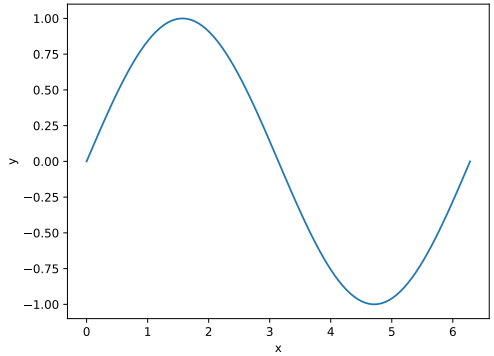
Научимся выбирать вид графика и масштаб осей, чтобы показывать зависимости, сравнивать наборы данных и отображать погрешности измерений.

Первый график в Matplotlib

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

x = np.linspace(0, 2 * np.pi, 100)
y = np.sin(x)

plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```

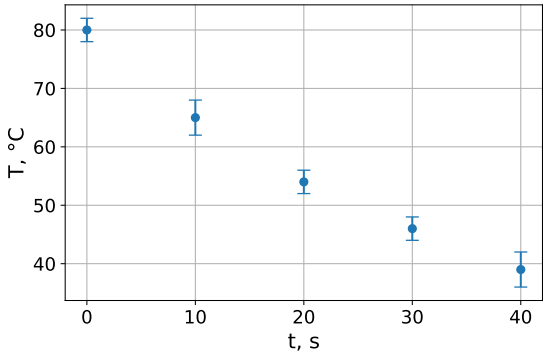


`plot(x, y)` соединяет точки линией.
Массивы x и y содержат по 100 значений.

Погрешности: errorbar

Учебный пример: температура и погрешность измерения.

```
t = np.array([0, 10, 20, 30, 40])
T = np.array([80, 65, 54, 46, 39])
dT = np.array([2, 3, 2, 2, 3])
plt.figure(figsize=(6, 4))
plt.errorbar(t, T, yerr=dT,
             fmt='o', capsize=5)
plt.xlabel('t, s', fontsize=16)
plt.ylabel('T, °C', fontsize=16)
plt.tick_params(labelsize=14)
plt.grid()
plt.tight_layout()
plt.show()
```



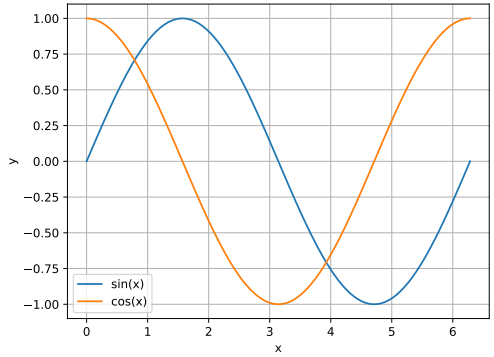
`yerr=dT` задаёт погрешности по вертикали.
Каждый отрезок идёт от `T-dT` до `T+dT`.

Несколько кривых и легенда

```
x = np.linspace(0, 2 * np.pi, 100)
y1 = np.sin(x)
y2 = np.cos(x)

plt.plot(x, y1, label='sin(x)')
plt.plot(x, y2, label='cos(x)')

plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid()
plt.show()
```

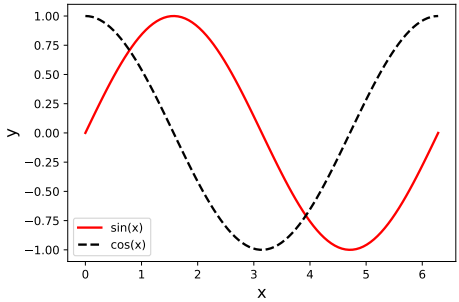


`label` задаёт подпись кривой.

`legend()` показывает легенду.

Оформление и сохранение графика

```
x = np.linspace(0, 2 * np.pi, 100)
plt.figure(figsize=(6, 4))
plt.plot(x, np.sin(x), 'r-', lw=2,
         label='sin(x)')
plt.plot(x, np.cos(x), 'k--', lw=2,
         label='cos(x)')
plt.xlabel('x', fontsize=14)
plt.ylabel('y', fontsize=14)
plt.legend()
plt.savefig('curves.png', dpi=300,
            bbox_inches='tight')
plt.show()
```

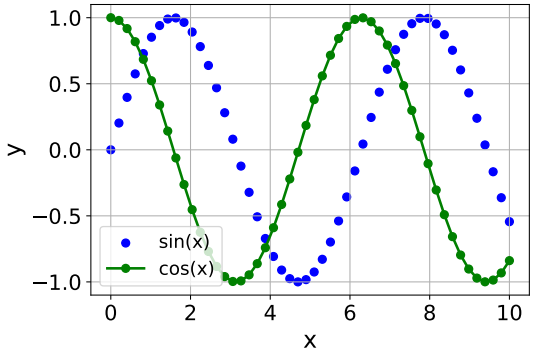


`figsize=(6, 4)` задаёт размер в дюймах, `lw` —толщину линии.

`savefig` сохраняет PNG с разрешением 300 точек на дюйм.

Точки и линии с маркерами

```
x = np.linspace(0, 10, 50)
y1 = np.sin(x)
y2 = np.cos(x)
plt.figure(figsize=(6, 4), dpi=100)
plt.scatter(x, y1, color='blue',
            label='sin(x)')
plt.plot(x, y2, color='green',
         marker='o', linestyle='-',
         linewidth=2, label='cos(x)')
plt.xlabel('x', fontsize=18)
plt.ylabel('y', fontsize=18)
plt.tick_params(labelsize=16)
plt.legend(fontsize=14)
plt.grid()
plt.show()
```



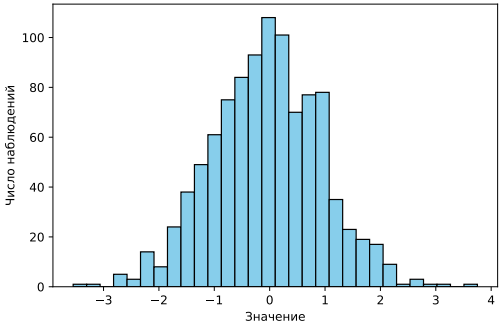
`scatter` показывает отдельные точки.

`plot(..., marker='o')` добавляет маркеры к линии.

Гистограмма

```
rng = np.random.default_rng(1)
data = rng.normal(0, 1, size=1000)

plt.figure(figsize=(6, 4))
plt.hist(data, bins=30,
         color='skyblue',
         edgecolor='black')
plt.xlabel('Значение')
plt.ylabel('Число наблюдений')
plt.tight_layout()
plt.show()
```



Тысяча значений из нормального распределения разбита на 30 интервалов. Высота столбца показывает, сколько значений попало в интервал.

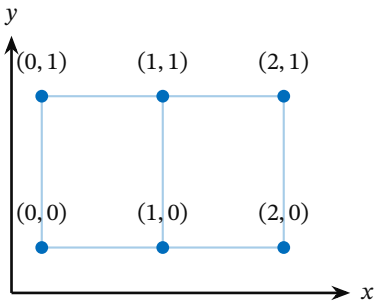
Координатная сетка: `meshgrid`

```
x = np.array([0, 1, 2])
y = np.array([0, 1])
X, Y = np.meshgrid(x, y)
```

Координаты узлов

$$X = \begin{pmatrix} 0 & 1 & 2 \\ 0 & 1 & 2 \end{pmatrix} \qquad Y = \begin{pmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

X хранит координаты по горизонтали, Y —по вертикали.

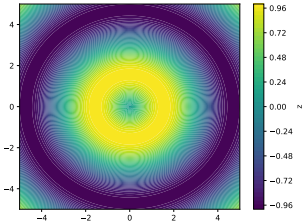


Три значения x и два значения y: шесть узлов сетки.

Одно поле на плоскости и в пространстве

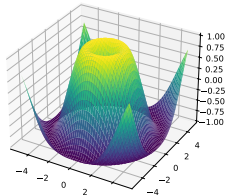
```
x = np.linspace(-5, 5, 100)
X, Y = np.meshgrid(x, x)
Z = np.sin(np.sqrt(X**2 + Y**2))
```

Цвет показывает значение Z




```
plt.figure()
c = plt.contourf(X, Y, Z, levels=50)
plt.colorbar(c, label='z')
plt.show()
```

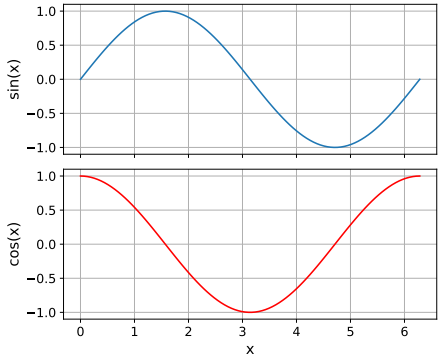
Высота показывает значение Z




```
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X, Y, Z, cmap='viridis')
plt.show()
```

Несколько графиков: subplots

```
x = np.linspace(0, 2 * np.pi, 100)
fig, axes = plt.subplots(2, 1,
    figsize=(6, 5), sharex=True)
axes[0].plot(x, np.sin(x))
axes[0].set_ylabel('sin(x)',
                   fontsize=14)
axes[1].plot(x, np.cos(x), color='red')
axes[1].set_ylabel('cos(x)',
                   fontsize=14)
axes[1].set_xlabel('x', fontsize=14)
for ax in axes:
    ax.tick_params(labelsize=12)
    ax.grid()
fig.tight_layout()
plt.show()
```

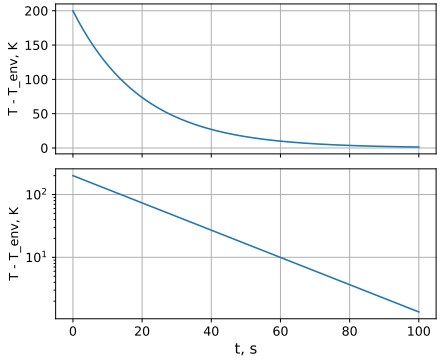


`fig` —весь рисунок.

`axes[0]` и `axes[1]` —два графика с общей осью x.

Логарифмическая шкала: semilogy

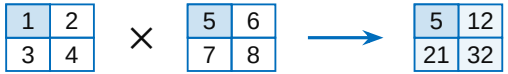
```
t = np.linspace(0, 100, 200)
dT = 200 * np.exp(-0.05 * t)
fig, axes = plt.subplots(2, 1,
    figsize=(6, 5), sharex=True)
axes[0].plot(t, dT)
axes[1].semilogy(t, dT)
for ax in axes:
    ax.set_ylabel('T - T_env, K',
                  fontsize=12)
    ax.tick_params(labelsize=12)
    ax.grid()
axes[1].set_xlabel('t, s', fontsize=14)
fig.tight_layout()
plt.show()
```



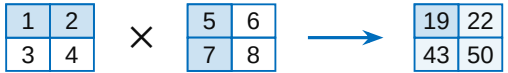
На логарифмической оси y экспонента выглядит как прямая. Значения по этой оси должны быть положительными.

Приложение. Умножение в NumPy и MATLAB

Поэлементное
NumPy: `A * B`
MATLAB: `A .* B`



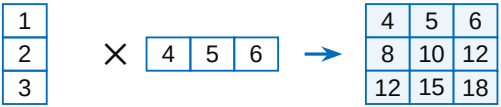
Матричное
NumPy: `A @ B`
MATLAB: `A * B`



Скалярное
NumPy: `np.dot(a, b)`
MATLAB: `dot(a, b)`



Внешнее
NumPy: `np.outer(a, b)`
MATLAB: `a.' * b`



a, b: в NumPy — одномерные массивы, в MATLAB — строки.

Приложение. Каталоги и файлы в Linux

В терминале Ubuntu (WSL) создайте рабочую папку и перейдите в неё.

```
mkdir -p ~/work/linux-demo
cd ~/work/linux-demo
```

Действие	Команда
Показать путь к текущей папке	`pwd`
Вывести подробный список, включая скрытые файлы	`ls -la`
Создать пустой файл	`touch notes.txt`
Сделать копию	`cp -i notes.txt copy.txt`
Переименовать копию	`mv -i copy.txt draft.txt`
Удалить копию с подтверждением	`rm -i draft.txt`

`cd ..` поднимает на один уровень, `cd ~` возвращает в домашнюю папку.

Приложение. Просмотр и поиск в Linux

Вернитесь в рабочую папку и создайте короткий журнал. Знак > записывает вывод заново, а >> добавляет строку в конец.

```
cd ~/work/linux-demo
printf 'INFO start\nERROR: no input\n' > run.log
printf 'INFO end\n' >> run.log
```

Действие	Команда
Прочитать файл целиком	`cat run.log`
Листать файл (клавиша q для выхода)	`less run.log`
Первые или последние две строки	`head -n 2 run.log` / `tail -n 2 run.log`
Найти файлы с расширением `.log`	`find . -type f -name '*.log'`
Найти ошибку и номер строки	`grep -in 'error' run.log`

Результат `grep`: `2:ERROR: no input`.

Итоги лекции

После лекции вы сможете самостоятельно

- **Подготовить среду для расчёта** и запустить программу с нужной версией Python и набором библиотек.
- **Перевести алгоритм в программу** с помощью переменных, условий, циклов и функций.
- **Обработать численные данные в NumPy,** выбирая элементы массива и операции с учётом его формы.
- **Представить результаты на графике,** подобрав вид графика и шкалы осей для сравнения величин.