БИБЛИОТЕКИ ЯЗЫКА PYTHON ДЛЯ КОМПЬЮТЕРНЫХ ВЫЧИСЛЕНИЙ И МОДЕЛИРОВАНИЯ

Библиотеки **NumPy, Matplotlib, SciPy** языка **Python**

Лобанов Алексей Владимирович
Главный специалист отдела разработки и внедрения АИС,
Ассистент кафедры информационных компьютерных технологий РХТУ им. Д.И. Менделеева

ТЕМЫ

- Менеджер пакетов pip
- Библиотека **numpy**
- Библиотека **matplotlib**
- Библиотека **scipy**

ПОЛЕЗНЫЕ РЕСУРСЫ

- Библиотека **numpy**
- Библиотека **numpy** (оф. сайт)
- Библиотека **matplotlib** (оф. сайт)
- Библиотека **matplotlib** (курс)
- Библиотека **scipy** (оф. сайт)

СТОРОННИЕ БИБЛИОТЕКИ

Библиотеки Python — это коллекции дополнительных модульных компонентов кода для «змеиного языка», заточенных под определенные тонкие задачи. Для управления ими необходимы специальные навыки, овладев которыми можно сделать программирование на «Питоне» значительно более эффективным.

Модули сторонних библиотек нужны для расширения функционала стандартной библиотеки Python. С помощью набора функций, который они предлагают, можно поэтапно работать над большими проектами и решать различные комплексные задачи.

Python с библиотеками

Python без них

МЕНЕДЖЕР ПАКЕТОВ PIP

pip — это система управления пакетами, которая используется для установки и управления программными пакетами, написанными на Python. Не все пакеты нужны в повседневной практике или отдельном проекте, да и места они занимают не мало. Для этих целей создан удаленный репозиторий модулей https://pypi.org/, в котором на сегодня имеется более 260 тыс. проектов на все случаи практики программирования. Вам не обязательно создавать код с нуля, так как под многие задачи уже имеется соответствующий пакет. Работа с этим хранилищем расширений осуществляется через команду pip.

Установка пакета

```
pip install numpy
```

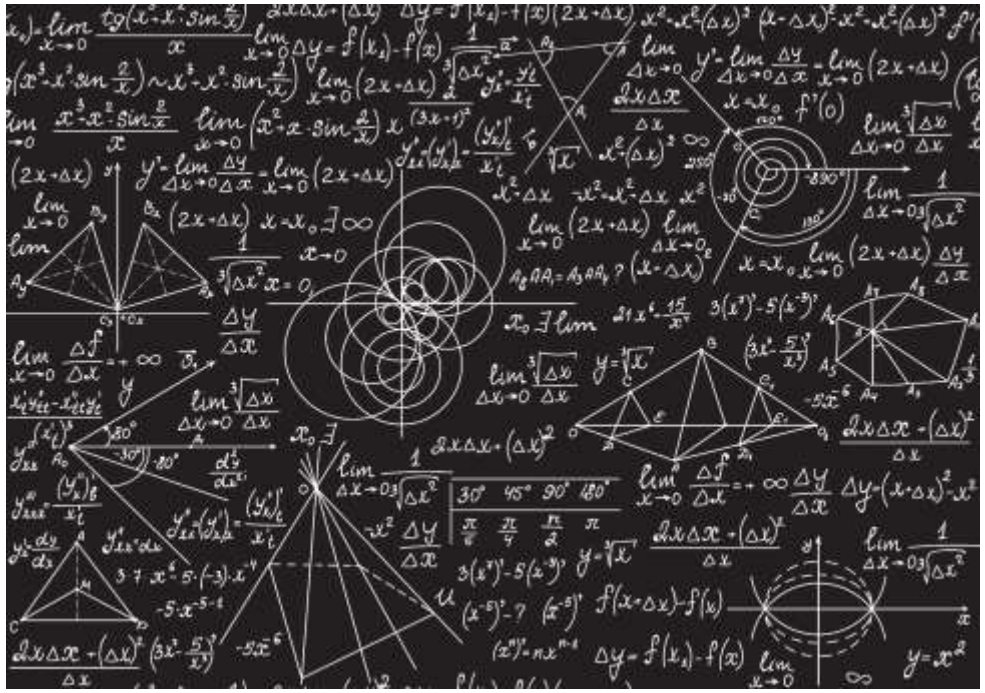
Удаление пакета

```
pip uninstall pyzipper
```

БИБЛИОТЕКА NUMPY

NumPy — это библиотека языка Python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых (и очень быстрых) математических функций для операций с этими массивами.

$$A = \begin{pmatrix} 3 & -1 & 1 \\ 5 & 1 & 2 \\ 1 & 1 & 2 \end{pmatrix}$$



БИБЛИОТЕКА NUMPY

Для создания массива в **numpy** используется функция **array**, кроме того, существует ряд функций, создающих специализированные массивы

```python
import numpy as np

a = np.array([1, 2, 3])

#Нулевая матрица
z = np.zeros([2, 2])

#Матрица единиц
z = np.ones([2, 2])

#Единичная матрица
b = np.eye(3)

#5 чисел в промежутке от 0 до 1
a = np.linspace(0, 1, 5)

#как range только вернет массив
b = np.arange(0, 1, 0.1)
```

БИБЛИОТЕКА NUMPY

Массивы **numpy** поддерживают ряд матричных операций и преобразований, таких как транспонирование, сложение, вычитание, матричное и поэлементное умножение. Помимо этого, существует модуль **linalg**, позволяющий решать различные задачки линейной алгебры.

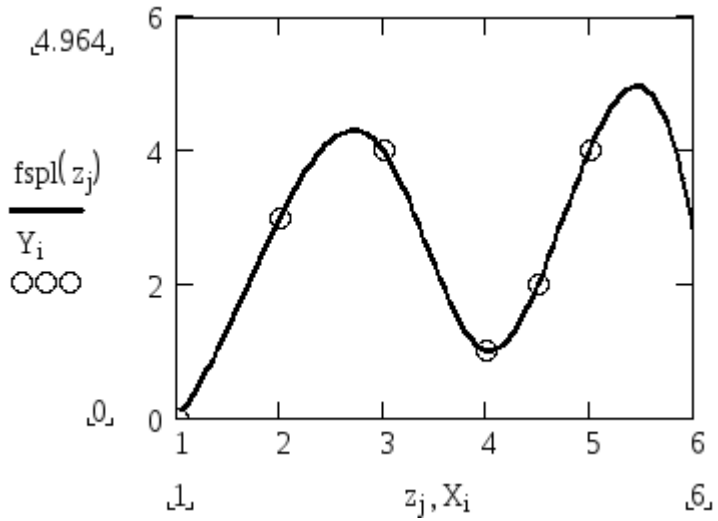
```
a = np.array([[0, 1, 2, 3],
[4, 5, 6, 7],
[8, 9, 10, 11]
])
print("a =\n", a)

#транспонирование
c = a.T
print("c =\n", c)

#поэлементные операции
a= np.array([[1, 2, 3],
[4, 5, 6]])
b = np.array([1, 2, 4])
c0 = a + a
print("c0 =\n", c0)
c1 = a + b
print("c1 =\n", c1)
c2 = a * b
print("c2 =\n", c2)

#матричное умножение
c3 = np.dot(a, b)
print("c3 =\n", c3)
c4 = a @ b
print("c4 =\n", c4)
```

```
a = np.array([[-4, -3, -2],
[-1, 0, -1],
[2, 3, 4]])
# Норма матрицы
norm = np.linalg.norm(a)
print("norm =", norm)
# Определитель матрицы
det = np.linalg.det(a)
print("det =", det)
# След матрицы
trace = np.trace(a)
print("trace =", trace)
# Обратная матрица
inv = np.linalg.inv(a)
print("a =\n", a)

a = np.array([[1, 2],
[3, 5]])
b = np.array([1, 2])
# Точное решение СЛАУ ax=b
s1 = np.linalg.solve(a, b)
print("s1 =", s1)
# Решения СЛАУ методом
# наименьших квадратов
s2 = np.linalg.lstsq(a, b)[0]
print("s2 =", s2)
```

БИБЛИОТЕКА NUMPY

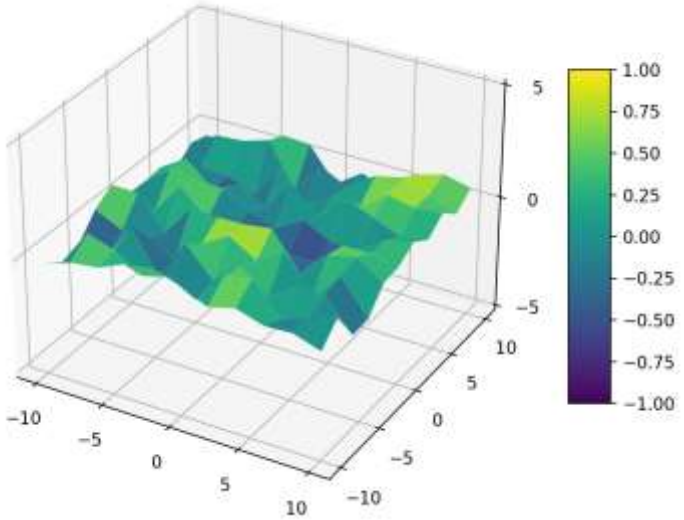
numpy позволяет без лишних усилий для программиста решать различные вычислительные задачи, связанные с векторами и матрицами, кроме того библиотеки имеет ряд встроенных функций, например функции для аппроксимации данных многочленом любой степени, при помощи функций **polyfit** и **polyval.**

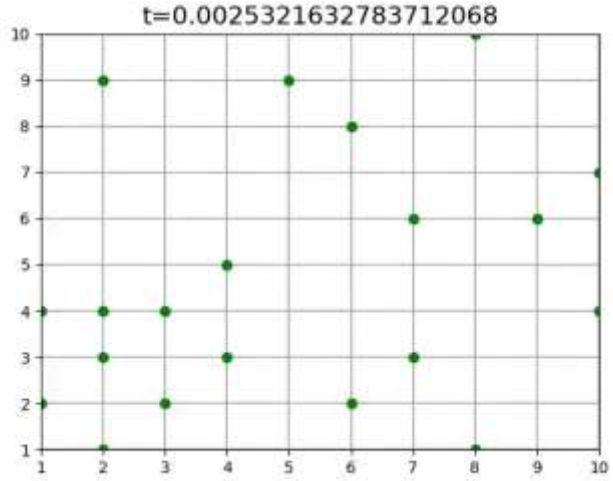
```
xs = np.array([0, 1, 2, 3, 4])
ys = np.array([1, 3, 4, 5, 7])

# Расчет коэффициентов (полином третьей степени)
p = np.polyfit(xs, ys, deg=3)
print(p)
# Получение расчетных значений
yc = np.polyval(p, xs)
print(yc)
```



БИБЛИОТЕКА MATPLOTLIB

MatPlotLib - Matrix Plotting Library, обширная библиотека предоставляющая функционал для реализации двумерной и трехмерной графики. Генерируемые изображения могут быть использованы в интерактивной графике, научных публикациях и пользовательских графических интерфейсах. MatPlotLib предоставляет несколько интерфейсов для построения графиков: функциональный и объектно-ориентированный. Функциональный подход является более простым и практически аналогичен MATLAB, а объектно-ориентированный подход предоставляет более гибкие возможности настройки получаемых графиков.

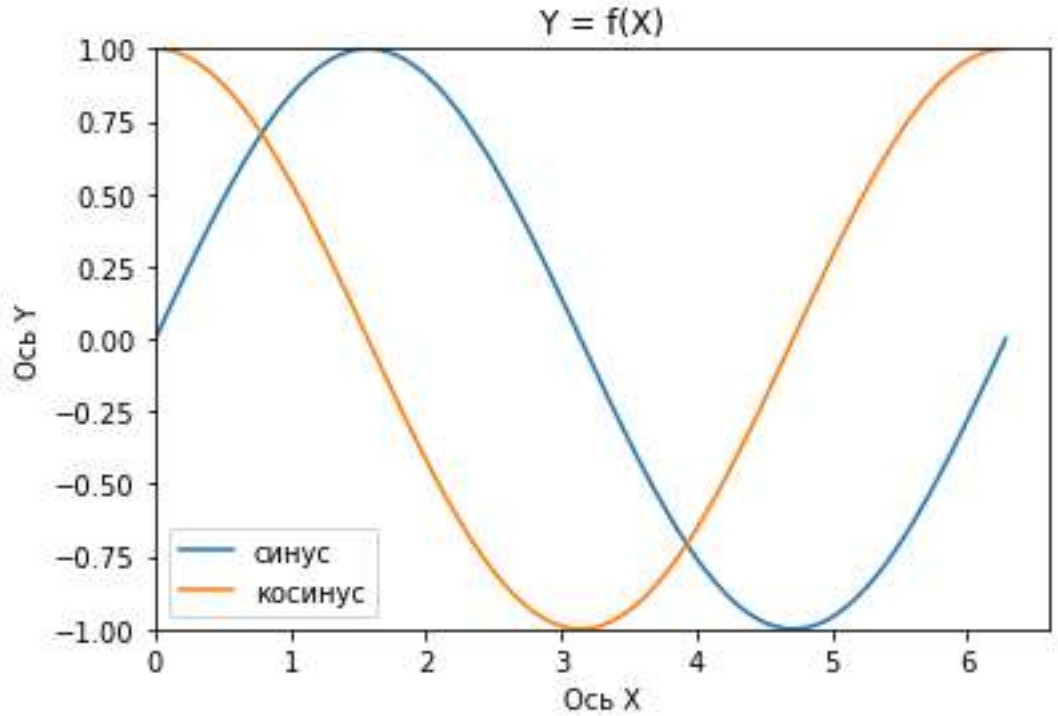




БИБЛИОТЕКА MATPLOTLIB

При построении графиков в блокнотах Jupyter, получившиеся изображения будут становиться частью блокнота, а при построении графиков с использованием файлов с разрешением .py они будут выводиться в интерактивном окне.

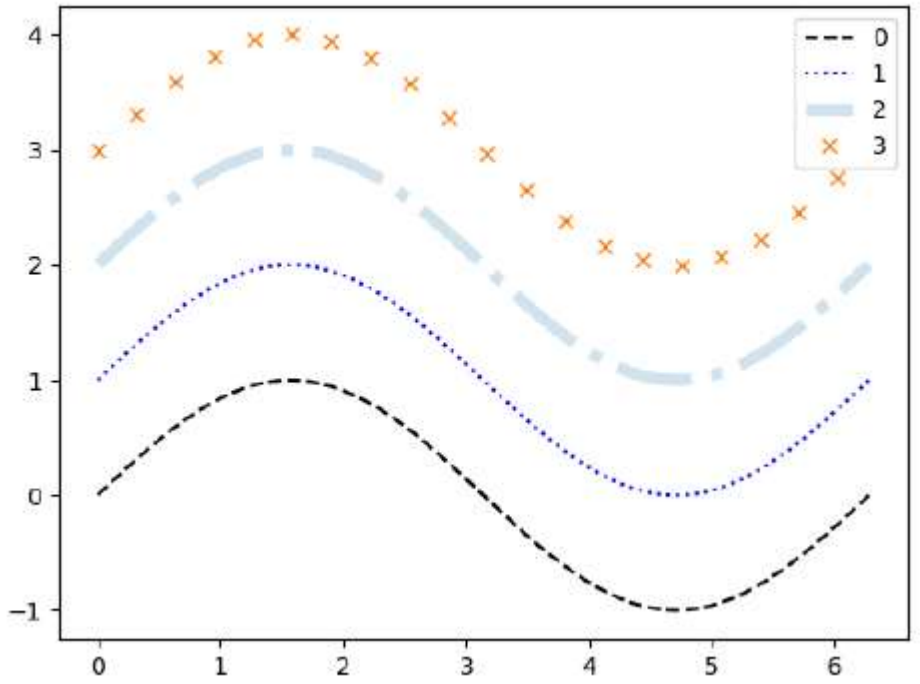
```
import numpy as np
import matplotlib.pyplot as plt

# Массив X'ов от 0 до 2 Пи
xs = np.linspace(0, 2*np.pi, 100)
# Рассчет Y'ов
ys = np.sin(xs)
# Построение линии
plt.plot(xs, ys, label='синус')

# Построение второй линии
ys2 = np.cos(xs)

plt.plot(xs, ys2, label='косинус')

# Подпись оси X
plt.xlabel("Ось X")
# Подпись оси Y
plt.ylabel("Ось Y")

# Заголовок
plt.title("Y = f(X)")

# Ограничение оси X
plt.xlim(left=0)
# Ограничение оси Y
plt.ylim(-1, 1)

# Отображение легенд
plt.legend()

# Отображение фигуры
plt.show()
```



БИБЛИОТЕКА MATPLOTLIB

Графики поддерживают различного рода форматирование, формат указывается как строка, содержащая определенный набор символов, который отвечает за стиль отображения данных.

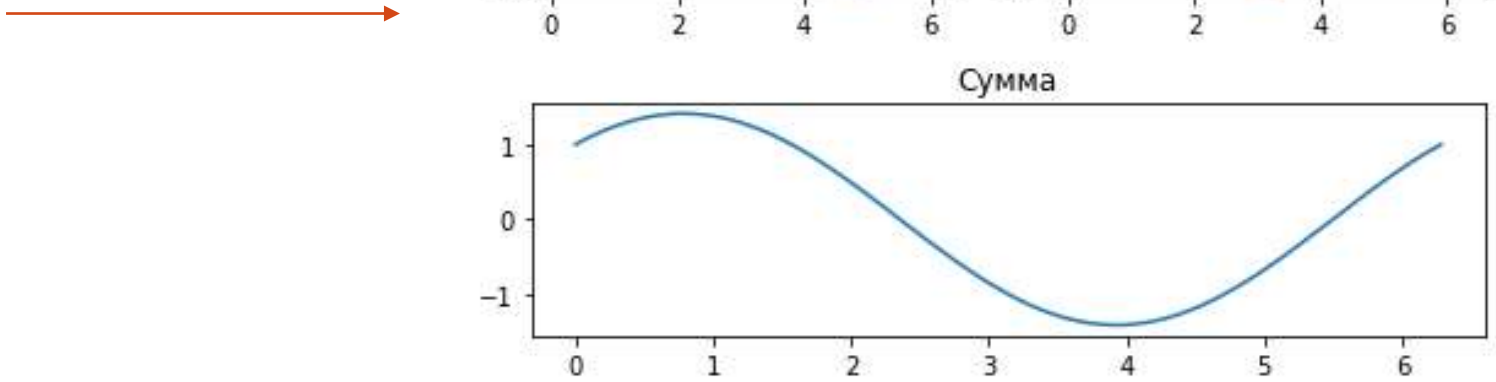
```
xs = np.linspace(0, 2*np.pi, 100)
s = np.sin(xs)
ys = [s, s+1, s+2, s+3]

# Черный штрих
plt.plot(xs, ys[0], "--k", label=0)
# Синий пунктир
plt.plot(xs, ys[1], c="blue",
    ls=":", label=1)
# Прозрачный штрих-пунктир
plt.plot(xs, ys[2], ls="-.",
    lw=3, alpha=0.5, label=2)
# Без линии с маркерами
plt.plot(xs, ys[3],
    ls="", marker="x",
    markevery=5, label=3)

plt.legend()
plt.show()
```



БИБЛИОТЕКА MATPLOTLIB

В рамках одного графика можно создавать несколько подграфиков при помощи функции **subplot.** Первые два числа – размеры матрицы для выстраивания графиков, третье – номер ячейки, где строится график.

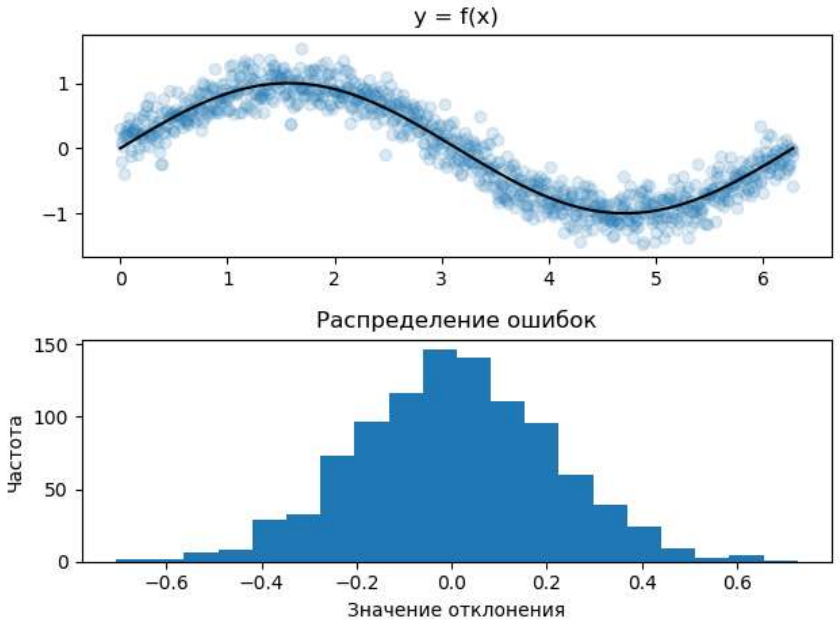
```
import numpy as np
import matplotlib.pyplot as plt

xs = np.linspace(0, 2*np.pi, 100)
ys1 = np.sin(xs)
ys2 = np.cos(xs)
# Создание первого подграфика
plt.subplot(221)
plt.plot(xs, ys1)
plt.title("Синус")
# Создание второго подграфика
plt.subplot(222)
plt.plot(xs, ys2)
plt.title("Косинус")
# Создание третьего подграфика
plt.subplot(212)
plt.plot(xs, ys1 + ys2)
plt.title("Сумма")
# Корректировка расстояний
# между подграфиками
plt.tight_layout()
plt.show()
```



БИБЛИОТЕКА MATPLOTLIB

Библиотека позволяет стоить графики различных типов, в том числе гистограммы и точечные диаграммы.

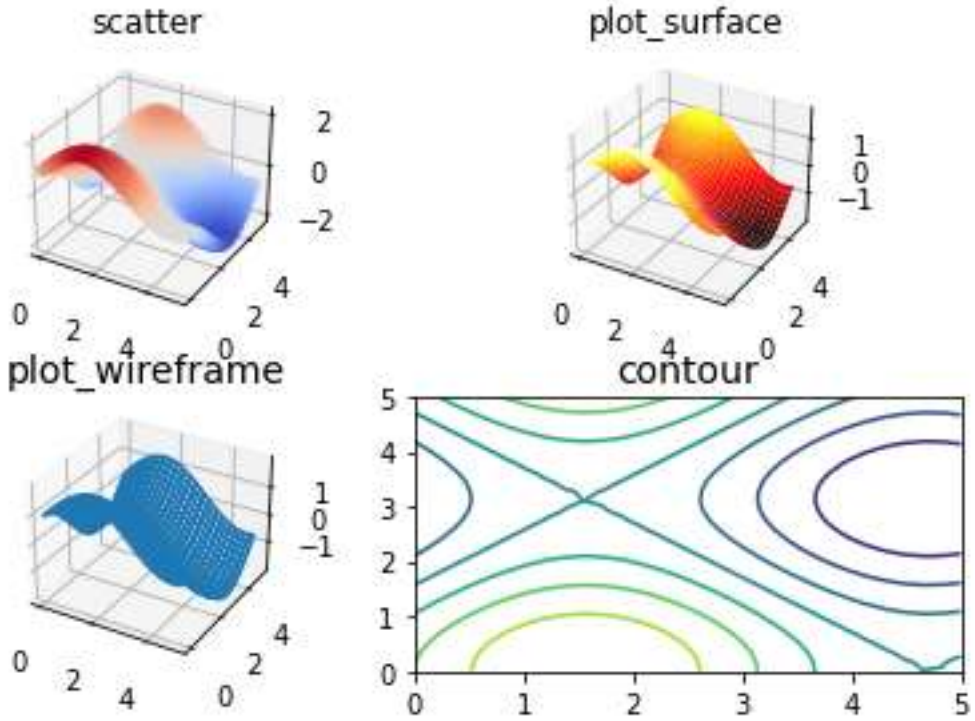
```python
xs = np.linspace(0, 2*np.pi, 1000)
ys = np.sin(xs)
# Создание псевдо-экспериментальных данных
yr = ys + np.random.randn(ys.size) / 5
# Расчет отклонений
err = ys - yr

plt.subplot(211)
plt.plot(xs, ys, c="black")
# Построение дискретных точек
plt.scatter(xs, yr, alpha=0.15)
plt.title("y = f(x)")

plt.subplot(212)
# Построение гистограммы
plt.hist(err, bins=20)
plt.title("Распределение ошибок")
plt.xlabel("Значение отклонения")
plt.ylabel("Частота")

plt.tight_layout()
plt.show()
```



БИБЛИОТЕКА MATPLOTLIB

Использование ООП открывает доступ к гораздо большему количеству настроек и модификаций, в том числе упрощает построение трехмерных графиков. Практически все примеры на официальном сайте MatPlotLib выполнены с применением ООП.

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

# Создание исходных данных
x = np.linspace(0, 5, 30)
# Создание координатной сетки
X, Y = np.meshgrid(x, x)
Z = np.sin(X) + np.cos(Y)
# Создание основной фигуры
fig = plt.figure()
# Добавление на фигуру первой 3D проекции
ax1 = fig.add_subplot(221, projection='3d')
ax1.scatter(X, Y, Z, c=Z.flatten(), cmap=mpl.cm.coolwarm)
ax1.set_title("scatter")
# Добавление на фигуру второй 3D проекции
ax2 = fig.add_subplot(222, projection='3d')
ax2.plot_surface(X, Y, Z, cmap=mpl.cm.hot)
ax2.set_title("plot_surface")
# Добавление на фигуру третьей 3D проекции
ax3 = fig.add_subplot(223, projection='3d')
ax3.plot_wireframe(X, Y, Z)
ax3.set_title("plot_wireframe", fontsize=14)

# Добавление на фигуру четвертой 3D проекции
ax4 = fig.add_subplot(224)
ax4.contour(X, Y, Z)
ax4.set_title("contour", fontsize=14)
plt.tight_layout()
plt.show()
```



БИБЛИОТЕКА SCIPY

SciPy — это библиотека Python с открытым исходным кодом, предназначенная для решения научных и математических проблем. Она построена на базе NumPy и позволяет управлять данными, а также визуализировать их с помощью разных высокоуровневых команд.

БИБЛИОТЕКА SCIPY (ИНТЕРПОЛЯЦИЯ)

Интерполяция полиномом

```
from scipy.interpolate import interp1d
import matplotlib.pyplot as plt
import numpy as np

xs = np.linspace(0, 2*np.pi, 1000)
ys = np.sin(xs)
x = np.linspace(0, 2*np.pi, 10)
y = np.sin(x)
fl = interp1d(x, y, kind="linear")
fq = interp1d(x, y, kind="quadratic")
fc = interp1d(x, y, kind="cubic")
plt.plot(xs, ys, label="Действительное")
plt.plot(x, y, "o", label="Дискретное")
plt.plot(xs, fl(xs), ls="-", label="Линейная")
plt.plot(xs, fq(xs), ls="-.", label="Квадратичная")
plt.plot(xs, fc(xs), ls=":", label="Кубическая")
plt.legend()
plt.show()
```

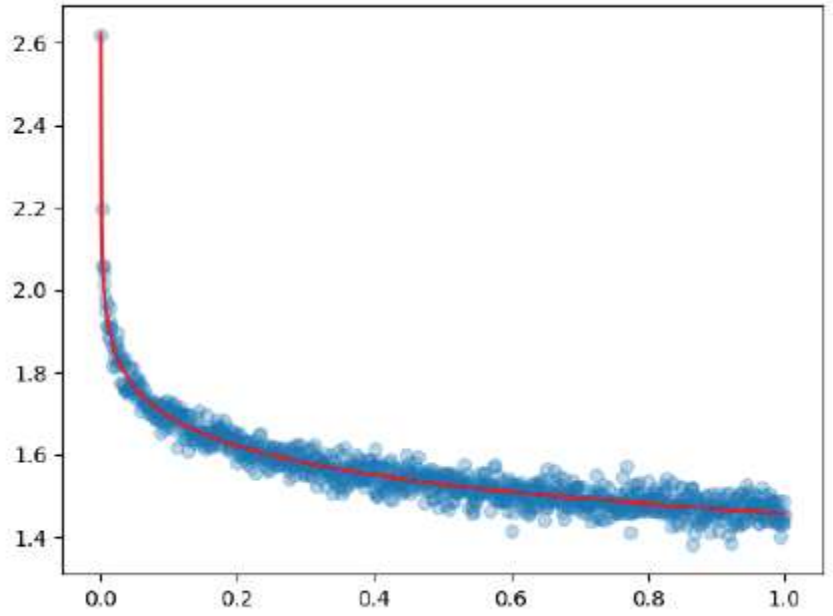
Интерполяция многочленом Лагранжа

```
from scipy.interpolate import lagrange
import matplotlib.pyplot as plt
import numpy as np

xs = np.linspace(0, 2*np.pi, 1000)
ys = np.sin(xs)
x = np.linspace(0, 2*np.pi, 10)
y = np.sin(x)
L = lagrange(x, y)
plt.plot(xs, ys, label="Действительное")
plt.plot(x, y, "o", label="Дискретное")
plt.plot(xs, L(xs), ls=":",
c="r", label="Лагранж")
plt.legend()
plt.show()
```

БИБЛИОТЕКА SCIPY (АППРОКСИМАЦИЯ)

Аппроксимация произвольной функцией

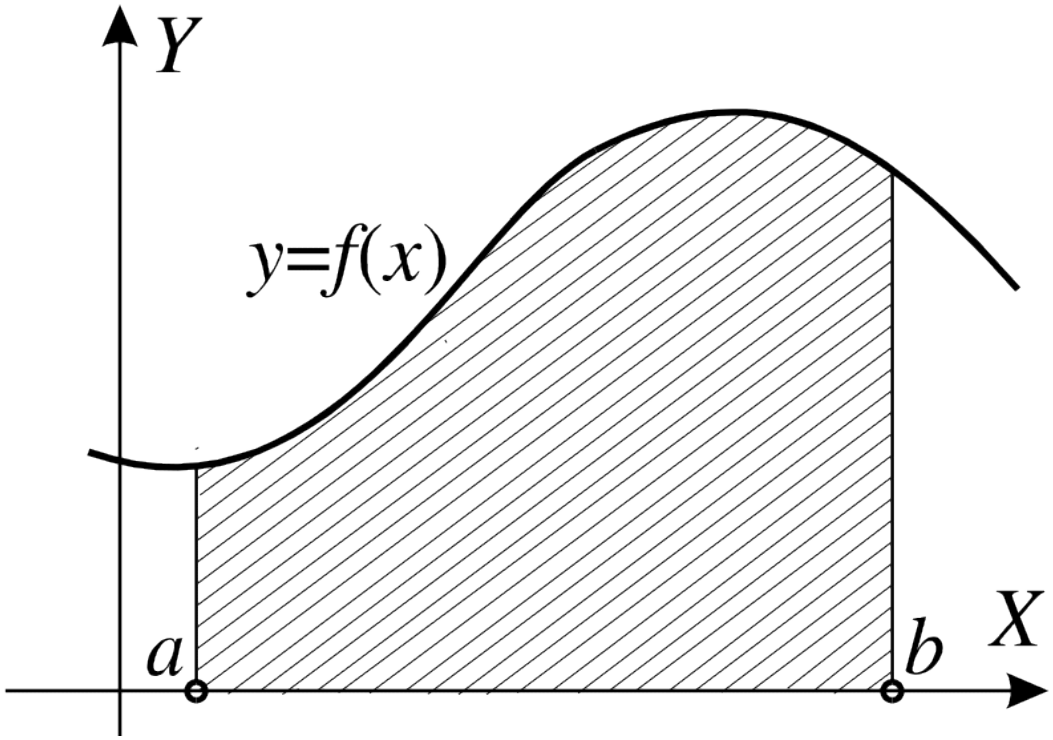
```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
import numpy as np

def eq(x, a, b, c):
    return a + b*np.log(1e-7 + x * c)
# Создание псевдо-экспериментальных данных
xs = np.linspace(0, 1, 1000)
err = np.random.randn(xs.size) / 40
ys = eq(xs, 1, -0.1, 0.01) + err
# Расчет коэффициентов
cs = curve_fit(eq, xs, ys)[0]
# Получение расчетных значений
yc = eq(xs, *cs)
plt.scatter(xs, ys, alpha=0.3)
plt.plot(xs, yc, c="r")
plt.show()
```



БИБЛИОТЕКА SCIPY (ИНТЕГРИРОВАНИЕ)

Определённый интеграл

```
import numpy as np
from scipy.integrate import (
quad, trapz, simps
)

# Задание целевой функции
f = lambda x: 2 - 0.5 * np.sin(x)
# Задание границ интегрирования
a, b = -np.pi, np.pi
# Расчет дисктретных значений
xs = np.linspace(a, b, 101)
ys = f(xs)
q = quad(f, a, b)
t = trapz(ys, xs)
s = simps(ys, xs)
print(q, t, s, sep="\n")
```



БИБЛИОТЕКА SCIPY (КОРЕНЬ НЕЛИНЕЙНОГО УРАВНЕНИЯ)

Корень уравнения

```
import numpy as np
from scipy.optimize import root

def f(x):
    return np.sin(x)
res = root(f, 3)
print(res.x, res.fun, sep="\n")
```

БИБЛИОТЕКА SCIPY (РЕШЕНИЕ СНАУ)

Корни СНАУ

```
import numpy as np
from scipy.optimize import root

def f(x):
    x, y = x
    return np.array([
        np.sin(x) - 2*y - 1.6,
        np.cos(y + 0.5) + x - 0.8
        ])


res = root(f, [0.5, 0.5])
print(res.x, res.fun, sep="\n")
```

БИБЛИОТЕКА SCIPY (ОПТИМИЗАЦИЯ)

Одномерная оптимизация

```
import numpy as np
from scipy.optimize import minimize

f = np.poly1d([
-0.01553, -0.09659, 0.1198,
1.148, 0.1749, -1.019
])

res = minimize(f, 0)
print(res.x, res.fun, sep="\n")
```

Многомерная оптимизация

```
import numpy as np
from scipy.optimize import minimize

def f(x):
    return np.sin(x[0]) + np.cos(x[1])

res = minimize(f, [0.5, 0.5])
print(res.x, res.fun, sep="\n")
```

БИБЛИОТЕКА SCIPY (ДИФФУРЫ)

Решение ОДУ

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

def f(t, y, k=1, r=1):
    dP_dt = r * y * (1 - y / k)
    return dP_dt

def log(t, p0=0.5, k=1, r=1):
    h1 = k*p0*np.exp(r*t)
    h2 = k+p0*(np.exp(r*t)-1)
    return h1 / h2

ts = [-6, 6]
y0 = [log(-6)]
t_eval = np.linspace(*ts, 100)
res = solve_ivp(f, ts, y0, t_eval=t_eval)
plt.plot(res.t, res.y[0, :], label="Расч")
plt.plot(res.t, log(res.t),
ls="-.", label="Действ")
plt.legend()
```

Решение систем ОДУ

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

def f(t, y):
    dy1_dt = 2 * y[0] - 5 * y[1] + 3
    dy2_dt = 5 * y[0] - 6 * y[1] + 1
    return [dy1_dt, dy2_dt]


ts = [0, 2]
y0 = [6, 5]
t_eval = np.linspace(*ts, 100)
res = solve_ivp(f, ts, y0, t_eval=t_eval)
plt.plot(res.t, res.y[0, :], label="Расч y1")
plt.plot(res.t, res.y[1, :], label="Расч y2")
```

ЗАДАНИЕ ДЛЯ САМОКОНТРОЛЯ

Исследовалась структура потока жидкости в аппарате колонного типа. Импульсным методом вводился индикатор на вход потока в аппарате, и измерялась его концентрация на выходе из аппарата через интервал времени delta t = 1мин. Ординаты экспериментальной кривой представлены в таблице. Начальное условие: при t=0 C(0)=0.

C	2,0109	3,34880	6,64870	16,16160	40,06250	91,7584

При помощи библиотеки numpy, создать программу, интерполирующую экспериментальные и вычислить значение концентрации в моменты времени t1=1,5мин t=2,5 мин.

ЗАДАНИЕ ДЛЯ САМОКОНТРОЛЯ

Построить кривые изменения концентрации компонентов приведённой реакции во временном интервале от 0 до 175:

$$A \xrightarrow{k1} R \xrightarrow{k2} B$$

Описать изменение концентраций можно при помощи системы дифф. уравнений:

$$\begin{cases} \dfrac{dc_a}{dt} = -k_1 * c_a \\ \dfrac{dc_r}{dt} = k_1 * c_a - k_2 * c_b \\ \dfrac{dc_b}{dt} = k_2 * c_r \end{cases}$$

В исходной смеси 5% R и 95% A. k1 = 0,05 k2=0,01.

СПАСИБО ЗА ВНИМАНИЕ!