

Задача 1. Основы numpy и matplotlib.

Морозов Олег Павлович

tg: o.morozov@g.nsu.ru

Физический факультет
Кафедра высшей математики
Новосибирский Государственный Университет

28.11.2024

План презентации

- 1 Необходимые основы numpy
- 2 Необходимые основы matplotlib
- 3 Комментарии к заданию №1
 - Шаг №1. Теория
 - Шаг №2. Реализация
 - Шаг 3. Результаты

Создание массивов **numpy**

Numpy - это библиотека для работы с многомерными массивами и выполнения численных вычислений. Она предоставляет высокопроизводительные структуры данных и функции для эффективной работы с массивами.

Создание массивов numpy

```
import numpy as np
vector = np.array([1, 2, 3])
arr = np.arange(0, 10, 1)
x = np.linspace(-np.pi, np.pi, 5)
```

Результат работы кода

Результат выполнения:

- vector: [1, 2, 3]
- arr: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- x: [-3.14, -1.57, 0.00, 1.57, 3.14]

Описание функций:

np.array - функция для создания массива.

np.arange - функция для создания массива с шагом.

np.linspace - функция для создания массива с равномерными интервалами.

Работа с массивами **numpy**

Действия над массивами

```
import numpy as np
norm = np.linalg.norm(vector)
arr_squared = np.power(arr, 2)
y1 = 2 * x + np.pi
y2 = np.sin(x)
def f(_x):
    return np.sin(_x) ** 2 + np.cos(_x) ** 2
y3 = f(x)
```

Результат работы кода

Результат выполнения:

- norm: 3.74
- arr_squared: [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
- y1: [-3.14, 0.00, 3.14, 6.28, 9.42]
- y2: [0.0, -1.0, 0.0, 1.0, 0.0]
- y3: [1., 1., 1., 1., 1.]

Описание функций:

np.linalg.norm - функция для вычисления нормы вектора.

np.power - функция для возведения элементов массива в степень.

np.sin - функция для вычисления синуса элементов массива.

f - функция созданная нами для проверки тождества.

Логические операции с массивами **numpy**, метод **where**

np.where(condition, True, False) - метод, который применяет условие **condition** ко всем элементам переданного ему массива и заполняет новый массив того же размера элементами **True** или **False** в зависимости от выполнения условия.

$$g(x) = \begin{cases} 0, & x \leq 0 \\ x^2, & x > 0 \end{cases}$$

Описание функции $g(x)$ с `np.where`

```
x = np.linspace(-1, 3, 100)
def g(_x):
    return np.where(_x>0., _x**2., 0.)
```

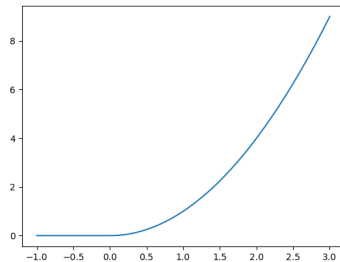


Рис. 1: Результат такого задания функции $g(x)$.

Интегрирование функций, метод **trapz**

np.trapz(f(x), x) - метод, который интегрирует функцию $f(x)$ по разбиению интервала на отрезки x , используя простейший метод численного интегрирования функций одной переменной - "*Метод трапеций*". Проинтегрируем функцию $g(x)$ на интервале $[-1, 3]$.

$$I_{exact} = \int_{-1}^3 g(x) dx = 9$$

Интегрирование $g(x)$ с np.trapz

```
x = np.linspace(-1, 3, 100)
Icalc = np.trapz(g(x), x)
```

Результат выполнения:

- I_{calc} : 9.000815

Тогда $error = |I_{exact} - I_{calc}| = 0.000815$ — ошибка интегрирования.

Создание и настройка графика в **matplotlib**

Matplotlib — это популярная библиотека для визуализации данных в Python. Она позволяет создавать широкий спектр графиков, от простых линейных графиков до сложных многоуровневых визуализаций. Рассмотрим, как был построен **рисунок 1** и модифицируем его до **рисунок 2**.



Рис. 2: Пример модифицированного графика $g(x)$.

Код реализующий **рисунок 2**

Настройка графика $g(x)$ (дополнительно в файле `example_matplotlib.py`)

```
import matplotlib.pyplot as plt
plt.plot(x, g(x), label='g(x)', color='darkgreen', linestyle='-')
plt.xlim(-1, 3)
plt.ylim(0, 9)
plt.grid(which='both', linestyle=':', linewidth=0.5)
plt.minorticks_on()
plt.xlabel("x")
plt.ylabel("y")
plt.title("График функции  $g(x)$ ")
plt.legend(fontsize=12)
plt.show()
```

Разделение рисунка на графики

Использование plt.subplots

```
fig, ax = plt.subplots(1,2)
ax[0].plot(x, g(x),
label='g(x)', color='darkgreen',
linestyle='-')
ax[1].plot(x, g(np.sqrt(x)),
label=r'g(sqrt(x))',
color='darkblue', linestyle=':')
```

Настройка каждой оси производится отдельно `ax[i]`.

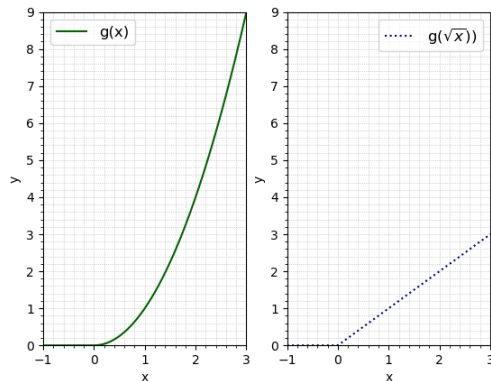


Рис. 3: Пример разделения графика на два.

Анимация в matplotlib

Создание анимации

```
from matplotlib.animation import  
FuncAnimation
```

```
fig, ax = plt.subplots()  
line, = ax.plot(x, g(x))
```

```
N = 100  
def update(frame): ->  
frames = np.linspace(1, N, 100)  
ani = FuncAnimation(fig, update,  
frames=frames, blit=True)
```

Функция update

```
def update(frame):  
    t = frame/N  
    y = g(x) * np.exp(-t)  
    line.set_data(x, y)  
    return line,
```

Результат

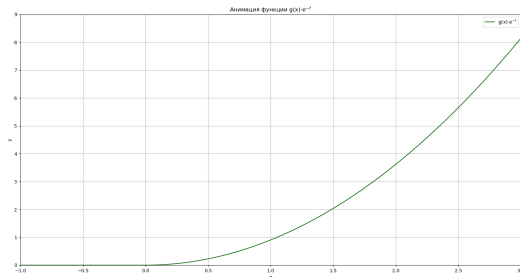


Рис. 4: График в момент времени t_0

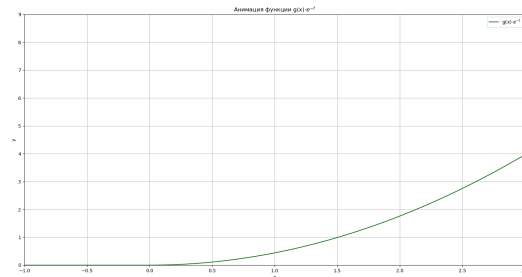


Рис. 5: График в момент времени t_1

Больше примером можно найти в учебнике [], а расширенный код из презентации на github [].

Условие к заданию №1

Создайте анимацию (при различном числе слагаемых N) из графиков частичных сумм рядов Фурье для функции

$$f(x) = \begin{cases} e^{-\frac{1}{1-x^2}}, & |x| < 1, \\ 0, & 1 \leq |x| \leq \pi, \end{cases}$$

и для функции

$$g(x) = f(x) + \varepsilon|x|, \quad |x| \leq \pi,$$

где ε — некоторое малое число.

Теория: поиск частичной суммы ряда Фурье.

Частичная сумма ряда Фурье функции $f(x)$ на интервале $[-\pi, \pi]$ может быть представлена в виде:

$$S_N(x) = \frac{a_0}{2} + \sum_{n=1}^N (a_n \cos(nx) + b_n \sin(nx)),$$

где коэффициенты a_n и b_n вычисляются по формулам:

$$a_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \cos(nx) dx,$$

$$b_n = \frac{1}{\pi} \int_{-\pi}^{\pi} f(x) \sin(nx) dx.$$

Реализация функций $f(x)$, $g(x)$.

Для реализации функции $f(x)$ рекомендую использовать `np.where`, чтобы поддержать работу массивов `numpy`.

Функции $f(x)$ и $g(x)$

```
def f(x):  
    return np.where(Условие, Если Да, Если Нет)  
  
eps = 0.1  
def g(x):  
    return 'Выражение от функции f(x)'
```

Функция f находится в области видимости функции g , поэтому передавать её не обязательно, но помните что есть возможность передавать функции друг другу, понадобится дальше.

Интегрирование.

Для реализации интегрирования рекомендую использовать `np.trapz` и предлагаю построить функцию интегрирования с весом ($1, \sin, \cos$) для того чтобы считать интегралы коэффициентов ряда Фурье.

Функция `weighted_integration`

```
x = np.linspace(-np.pi, np.pi, 100)
def weighted_integration(func, weight):
    return np.trapz()
I = C * weighted_integration(f(x), np.sin(x))
```

Коэффициент C и аргумент у \sin или \cos подбирается согласно теории. Значение 100 в определении x влияет на точность вычисления интегралов.

Поиск частичной суммы ряда Фурье.

Реализуем функцию для поиска N-ой частичной суммы ряда Фурье функции `func`.

Функция `fourier_series`

```
def fourier_series(x, func, N):  
    a0 = weighted_integration(func(x), 1)  
    result = a0 / 2  
    for n in range(1, N + 1):  
        an = C*weighted_integration(func(x), np.cos())  
        bn = C*weighted_integration(func(x), np.sin())  
        result += an * np.cos() + bn * np.sin()  
    return result
```

Функция возвращает значение частичной суммы в каждой точке. Важно то, что мы не считаем полную частичную сумму для каждой точки, а находим общие коэффициенты и подставляем сумму значения `x` - это производительнее.

Создание анимации

В функции `update` мы в качестве переменной `y` для обновления `line` берем значение N -ой частичной суммы ряда Фурье, можно сопоставить `frame = N`.

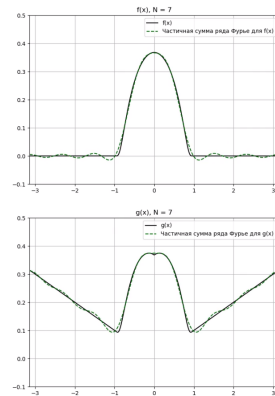
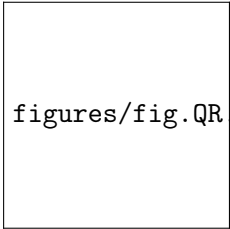


Рис. 6: Пример результата для задания №1.

Задача 1. Основы numpy и matplotlib.

Морозов Олег Павлович

o.morozov@g.nsu.ru



figures/fig.QR.png