Оглавление

[Сегмент 1: Импорт библиотек 2](#_Toc210301030)

[Сегмент 2: Генерация и сохранение данных 4](#_Toc210301031)

[Сегмент 3: Чтение данных и построение 2D-графиков 8](#_Toc210301032)

[Сегмент 4: Вывод статистики по столбца 13](#_Toc210301033)

[Сегмент 5: Фильтрация данных и сохранение в CSV 16](#_Toc210301034)

[Сегмент 6: Построение 3D-графика 19](#_Toc210301035)

# Сегмент 1: Импорт библиотек

python

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from mpl\_toolkits import mplot3d

**Строка 1: import pandas as pd**

* **Что это?** Это инструкция импорта библиотеки pandas.
* **Что делает?** Загружает библиотеку для работы с данными в память Python и делает её доступной под коротким псевдонимом pd.
* **Как работает под капотом?**
  1. Интерпретатор Python находит в своей среде установленную библиотеку pandas.
  2. Он загружает её основной модуль и выполняет весь код инициализации внутри него.
  3. Ключевое слово as создаёт ссылку на эту загруженную библиотеку с новым именем pd. Это стандартное соглашение в сообществе, которое экономит время набора и делает код чище. Вместо pandas.DataFrame мы теперь можем писать pd.DataFrame.

**Строка 2: import numpy as np**

* **Что это?** Импорт библиотеки numpy с псевдонимом np.
* **Что делает?** Делает доступными мощные инструменты для математических вычислений, особенно для работы с многомерными массивами.
* **Как работает под капотом?** Аналогично pandas. Загружается библиотека, и мы получаем доступ ко всем её функциям (например, numpy.linspace) через короткое имя np.linspace. NumPy является фундаментальной зависимостью для pandas, многие внутренние структуры pandas построены на массивах numpy.

**Строка 3: import matplotlib.pyplot as plt**

* **Что это?** Импорт модуля pyplot из библиотеки matplotlib с псевдонимом plt.
* **Что делает?** pyplot — это интерфейс, предоставляющий набор команд для построения графиков в стиле MATLAB (т.е. процедурно, "нарисуй линию", "добавь подпись" и т.д.).
* **Как работает под капотом?** matplotlib — это обширная библиотека. Модуль pyplot является её "удобным" фасадом. При импорте он может в фоновом режиме настраивать связь с бэкендом для отрисовки (например, с Tkinter, Qt и др.), который будет выводить графики в окна или в ячейки Jupyter Notebook. Псевдоним plt — ещё одно стандартное соглашение.

**Строка 4: from mpl\_toolkits import mplot3d**

* **Что это?** Это инструкция импорта конкретного модуля mplot3d из пакета mpl\_toolkits.
* **Что делает?** Загружает инструментарий для создания 3D-графиков в matplotlib.
* **Как работает под капотом?**
  + from ... import ... загружает не весь родительский пакет, а только указанный модуль. Это означает, что мы можем напрямую обращаться к его содержимому.
  + Модуль mplot3d добавляет в matplotlib новые классы и функции, такие как Axes3D, которые необходимы для задания 3D-осей на рисунке. Без этого импорта команда plt.axes(projection='3d') из 5-го пункта задания не сработала бы.

**Итог по сегменту 1:** Этот блок кода не производит видимого вывода, но он является критически важной подготовкой. Он загружает и инициализирует четыре ключевых компонента, которые будут использоваться для генерации данных, их обработки и визуализации. Теперь в нашем распоряжении находятся pd, np, plt и возможности для 3D-графиков.

# Сегмент 2: Генерация и сохранение данных

python

print("1. Генерация CSV-файла.")

* **Что это?** Оператор вывода на печать.
* **Что делает?** Выводит в консоль (или в ячейку Jupyter Notebook) строку-заголовок, чтобы визуально отделить этап генерации файла в общем потоке вывода.
* **Как работает под капотом?** Функция print() вызывает системный вызов, чтобы передать текстовую строку стандартному потоку вывода (stdout), который затем отображается средой выполнения.

python

n\_points = 500

* **Что это?** Присваивание значения переменной.
* **Что делает?** Создаёт переменную n\_points и сохраняет в неё целое число 500. Эта переменная будет определять количество строк (точек данных) в нашем наборе.
* **Как работает под капотом?** Интерпретатор Python создаёт объект типа int со значением 500 и связывает с ним имя n\_points в текущей области видимости.

python

x1 = np.linspace(-2\*np.pi, 2\*np.pi, n\_points)

* **Что это?** Создание первого массива входных данных с помощью функции NumPy linspace.
* **Что делает?** Генерирует массив из n\_points (500) равномерно распределённых чисел в диапазоне от -2 \* π до 2 \* π.
  + -2\*np.pi: Вычисляет значение нижней границы диапазона (примерно -6.283). np.pi — это константа из библиотеки NumPy.
  + 2\*np.pi: Вычисляет значение верхней границы диапазона (примерно 6.283).
  + n\_points: Количество генерируемых точек.
* **Как работает под капотом?** Функция np.linspace создаёт в памяти одномерный массив NumPy. Элементы этого массива вычисляются по формуле: start + (stop - start) \* i / (num - 1). Это гарантирует, что первое число будет равно start, а последнее — stop.

python

x2 = np.linspace(-np.pi, np.pi, n\_points)

* **Что это?** Создание второго массива входных данных.
* **Что делает?** Аналогично предыдущей строке, генерирует массив из 500 чисел в диапазоне от -π до π.
* **Как работает под капотом?** Происходит тот же процесс, что и для x1. Важно отметить, что на этом этапе x1 и x2 — это два *независимых* одномерных массива. Они ещё не образуют сетку.

python

y = np.cos(x1 + x2)

* **Что это?** Вычисление значений целевой функции согласно варианту (y = cos(x₁ + x₂)).
* **Что делает?** Вычисляет косинус от поэлементной суммы массивов x1 и x2.
* **Как работает под капотом?** Это демонстрация **векторизованных операций** в NumPy.
  1. x1 + x2: NumPy складывает два массива *поэлементно*. Первый элемент x1 складывается с первым элементом x2, второй со вторым и так далее. Результатом является новый массив той же длины (500 элементов).
  2. np.cos(): Функция косинуса применяется к *каждому элементу* результирующего массива суммы. Снова возвращается массив из 500 элементов.
  3. Это происходит очень быстро, без использования медленных циклов на Python, так как вычисления выполняются в скомпилированном коде на C/C++ под капотом NumPy.

python

data = pd.DataFrame({

'x\_1': x1,

'x\_2': x2,

'y': y

})

* **Что это?** Создание структуры данных DataFrame из словаря.
* **Что делает?** Объединяет три одномерных массива (x1, x2, y) в единую табличную структуру с именованными столбцами.
* **Как работает под капотом?**
  1. Сначала создаётся словарь Python: {'x\_1': x1, 'x\_2': x2, 'y': y}. Ключи словаря становятся названиями столбцов, а значения (наши массивы NumPy) — данными в этих столбцах.
  2. Конструктор класса pd.DataFrame принимает этот словарь и создаёт объект DataFrame. Каждый столбец в DataFrame по сути является массивом NumPy (pd.Series), что обеспечивает высокую производительность.
  3. Индекс (метки строк) по умолчанию устанавливается в RangeIndex от 0 до 499.

python

data.to\_csv('lab2\_data.csv', index=False)

* **Что это?** Метод для сохранения DataFrame в файл формата CSV (Comma-Separated Values).
* **Что делает?** Записывает содержимое DataFrame data в файл с именем 'lab2\_data.csv' в текущей директории. Параметр index=False указывает, что записывать индекс строк (столбец с числами 0,1,2,...) в файл не нужно.
* **Как работает под капотом?** Pandas проходит по всем строкам и столбцам DataFrame, преобразует данные в строковый формат, разделяет их запятыми и записывает получившиеся строки в файл. Первой строкой записываются названия столбцов (заголовок).

python

print("Файл 'lab2\_data.csv' успешно создан!")

print(f"Размер данных: {data.shape}\n")

* **Что это?** Вывод информации о результате операции.
* **Что делает?**
  + Первая строка сообщает об успешном создании файла.
  + Вторая строка использует f-строку для подстановки значения атрибута data.shape.
* **Как работает под капотом?**
  + data.shape — это атрибут DataFrame, возвращающий кортеж (количество\_строк, количество\_столбцов). В нашем случае — (500, 3).
  + F-строка f"..." вычисляет выражение внутри фигурных скобок {data.shape}, преобразует его в строку и вставляет в общий текст.
  + Символ \n в конце добавляет пустую строку для визуального отделения от следующего вывода.

**Итог по сегменту 2:** Мы создали три отдельных одномерных массива, вычислили по ним значения функции, упаковали всё в таблицу (DataFrame) и сохранили эту таблицу на диск. Ключевые технологии здесь — **векторизованные вычисления NumPy** и **работа с табличными данными в Pandas**.

# Сегмент 3: Чтение данных и построение 2D-графиков

python

print("2. Построение графиков.")

* **Что это?** Заголовок для второго этапа работы.
* **Что делает?** Информирует пользователя о том, что начинается визуализация данных.

python

data = pd.read\_csv('lab2\_data.csv')

* **Что это?** Чтение данных из CSV-файла обратно в DataFrame.
* **Что делает?** Загружает таблицу из файла 'lab2\_data.csv', который мы создали на предыдущем шаге, в переменную data.
* **Как работает под капотом?** Pandas парсит файл, распознаёт, что первая строка — это заголовки столбцов, и преобразует последующие строки в данные соответствующих типов (в данном случае, все столбцы будут иметь тип float64). Создаётся новый объект DataFrame.

python

plt.figure(figsize=(15, 5))

* **Что это?** Создание новой фигуры (окна для графиков) в matplotlib.
* **Что делает?** Инициализирует область для построения графиков с заданным размером 15 (ширина в дюймах) на 5 (высота в дюймах). Это широкий и невысокий холст, идеально подходящий для размещения двух графиков рядом.
* **Как работает под капотом?** Эта функция создаёт объект Figure, который является контейнером верхнего уровня для всех элементов визуализации.

**Левый график: y(x\_1) при постоянном x\_2**

python

plt.subplot(1, 2, 1)

* **Что это?** Создание первой подобласти (суб-графика) на фигуре.
* **Что делает?** Разбивает фигуру на сетку размером 1 строка и 2 столбца и делает первую ячейку этой сетки (индекс 1) активной. Все последующие команды plt.\* будут применяться к этой конкретной области.
* **Как работает под капотом?** Создаётся объект Axes (оси) и помещается в указанную позицию сетки. Индексация идёт слева направо и сверху вниз, начиная с 1.

python

x2\_const = data['x\_2'].mean()

* **Что это?** Вычисление константного значения для x\_2.
* **Что делает?** Берёт столбец 'x\_2' из DataFrame data и вычисляет его среднее арифметическое значение. Это значение будет использоваться как постоянное для всего первого графика.
* **Как работает под капотом?** Метод .mean() выполняет векторизованное вычисление среднего значения по всему массиву данных в столбце.

python

x1\_for\_plot = np.linspace(data['x\_1'].min(), data['x\_1'].max(), n\_points)

* **Что это?** Создание нового, плотного массива значений для x\_1.
* **Что делает?** Генерирует новый массив из 500 точек, равномерно распределённых от минимального до максимального значения x\_1, которые есть в наших данных.
* **Как работает под капотом?** data['x\_1'].min() и data['x\_1'].max() — это методы, которые находят минимальный и максимальный элементы в столбце. np.linspace создаёт плавный диапазон для красивого построения графика.

python

y\_for\_plot\_x1 = np.cos(x1\_for\_plot + x2\_const)

* **Что это?** Расчёт значений y для левого графика.
* **Что делает?** Вычисляет функцию y = cos(x1 + x2\_const) для нового плотного массива x1\_for\_plot и постоянного значения x2\_const.
* **Как работает под капотом?** Здесь используется **векторизованная операция** и **трансляция (broadcasting)**. Скаляр x2\_const мысленно "растягивается" до размеров массива x1\_for\_plot, и сложение происходит поэлементно.

python

plt.scatter(x1\_for\_plot, y\_for\_plot\_x1, alpha=0.7, color='blue', s=1)

* **Что это?** Построение точечного графика (диаграммы рассеяния).
* **Что делает?** Отображает точки с координатами (x1\_for\_plot, y\_for\_plot\_x1).
  + alpha=0.7: Задаёт полупрозрачность точек (0 — полностью прозрачные, 1 — полностью непрозрачные). Помогает увидеть плотность точек.
  + color='blue': Задаёт синий цвет точек.
  + s=1: Задаёт размер каждой точки в 1 пиксель. Так как точек много, они сливаются в плавную линию.
* **Как работает под капотом?** Функция scatter перебирает пары координат и отрисовывает каждую точку как отдельный графический примитив на активных осях.

python

plt.xlabel('x\_1')

plt.ylabel('y')

plt.title(f'y(x\_1) при x\_2 = {x2\_const:.2f} (константа)')

plt.grid(True, alpha=0.3)

* **Что это?** Настройка внешнего вида активных осей (левого графика).
* **Что делает?**
  + xlabel/ylabel: Добавляет подписи к осям X и Y.
  + title: Добавляет заголовок графика. F-строка подставляет вычисленное значение x2\_const, форматируя его до двух знаков после запятой (:.2f).
  + grid: Включает отображение сетки. Параметр alpha=0.3 делает сетку полупрозрачной, чтобы она не отвлекала от основных данных.

**Правый график: y(x\_2) при постоянном x\_1**

python

plt.subplot(1, 2, 2)

* **Что это?** Переключение на вторую подобласть.
* **Что делает?** Активирует вторую ячейку в сетке 1x2. Все последующие команды plt.\* теперь применяются к правому графику.

*(Далее логика полностью аналогична левому графику, но с заменой ролей*x\_1*и*x\_2*)*

python

x1\_const = data['x\_1'].mean()

x2\_for\_plot = np.linspace(data['x\_2'].min(), data['x\_2'].max(), n\_points)

y\_for\_plot\_x2 = np.cos(x1\_const + x2\_for\_plot)

plt.scatter(x2\_for\_plot, y\_for\_plot\_x2, alpha=0.7, color='red', s=1)

plt.xlabel('x\_2')

plt.ylabel('y')

plt.title(f'y(x\_2) при x\_1 = {x1\_const:.2f} (константа)')

plt.grid(True, alpha=0.3)

**Финальная настройка и отображение**

python

plt.tight\_layout()

* **Что это?** Функция для автоматической компоновки графиков.
* **Что делает?** Подстраивает отступы между суб-графиками и вокруг фигуры так, чтобы подписи, заголовки и метки осей не перекрывались.
* **Как работает под капотом?** Алгоритм анализирует все элементы на фигуре (заголовки, метки, тики) и вычисляет оптимальные поля.

python

plt.show()

* **Что это?** Команда для визуализации фигуры.
* **Что делает?** Отображает созданную фигуру со всеми её элементами (в нашем случае — два графика) в отдельном окне (если среда выполнения поддерживает это) или прямо в ячейке Jupyter Notebook.
* **Как работает под капотом?** Matplotlib передаёт отрисованную фигуру бэкенду по умолчанию (например, TkAgg, Qt5Agg), который управляет GUI-окном.

**Итог по сегменту 3:** Мы загрузили данные из файла и построили два отдельных 2D-графика, чтобы проанализировать влияние каждой переменной на функцию в отдельности, фиксируя вторую переменную на её среднем значении. Ключевые концепции — **суб-графики (subplots)** и **векторизованные вычисления с broadcasting**.

# Сегмент 4: Вывод статистики по столбца

python

print("3. Статистика по столбцам:")

* **Что это?** Заголовок для третьего этапа работы.
* **Что делает?** Информирует пользователя о том, что сейчас будет выведена статистическая информация.

python

for column in data.columns:

* **Что это?** Начало цикла for.
* **Что делает?** Организует последовательный перебор всех столбцов в DataFrame data. На каждой итерации цикла имя текущего столбца присваивается переменной column.
* **Как работает под капотом?**
  1. data.columns — это атрибут DataFrame, который возвращает объект Index, содержащий все названия столбцов (в нашем случае: ['x\_1', 'x\_2', 'y']).
  2. Цикл for итерируется по этому объекту. На первой итерации column = 'x\_1', на второй — 'x\_2', на третьей — 'y'.

python

print(f"Столбец {column}:")

* **Что это?** Вывод названия текущего столбца.
* **Что делает?** Печатает строку, например, "Столбец x\_1:". Это заголовок для блока статистики по конкретному столбцу.
* **Как работает под капотом?** F-строка подставляет значение переменной column в отображаемый текст.

python

print(f"- Среднее: {data[column].mean():.4f}")

* **Что это?** Вычисление и вывод среднего арифметического значения для текущего столбца.
* **Что делает?**
  1. data[column] — это обращение к столбцу DataFrame по его имени. Возвращается объект типа Series (фактически, один столбец таблицы).
  2. .mean() — метод, который применяется к этой серии данных и вычисляет среднее значение по всем её элементам.
  3. :.4f — спецификатор формата внутри f-строки. Он говорит: "отформатируй полученное число как floating-point (число с плавающей запятой) и выведи ровно 4 знака после запятой".
* **Как работает под капотом?** Метод .mean() делегирует вычисление нижележащему массиву NumPy. Для числовых данных это быстрая векторизованная операция. Pandas игнорирует пропущенные значения (NaN) при расчёте, но в нашем чистом наборе данных их нет.

python

print(f"- Минимальное: {data[column].min():.4f}")

* **Что это?** Вычисление и вывод минимального значения в столбце.
* **Что делает?** Аналогично предыдущему пункту. Метод .min() проходит по всем элементам столбца и находит наименьший.
* **Как работает под капотом?** Это также оптимизированная операция, выполняемая на уровне NumPy.

python

print(f"- Максимальное: {data[column].max():.4f}\n")

* **Что это?** Вычисление и вывод максимального значения в столбце.
* **Что делает?** Метод .max() находит наибольший элемент в столбце. Символ \n в конце строки добавляет пустую строку после вывода статистики для каждого столбца, визуально отделяя их друг от друга.
* **Как работает под капотом?** Аналогично .min().

**Итог по сегменту 4:** Этот код представляет собой элегантный и эффективный способ получить сводную статистику по всем столбцам таблицы данных. Он демонстрирует:

1. **Итерацию по структуре данных DataFrame** с помощью for column in data.columns.
2. **Использование встроенных статистических методов Pandas** (.mean(), .min(), .max()), которые являются обёртками над высокопроизводительными функциями NumPy.
3. **Форматированный вывод** для улучшения читабельности результатов.

В результате работы этого сегмента в консоли мы увидим аккуратно отформатированную таблицу со средними, минимальными и максимальными значениями для x\_1, x\_2 и y.

# Сегмент 5: Фильтрация данных и сохранение в CSV

python

print("4. Сохранение отфильтрованных данных.")

* **Что это?** Заголовок для четвёртого этапа работы.
* **Что делает?** Объявляет о начале этапа фильтрации данных.

python

mean\_x1 = data['x\_1'].mean()

mean\_x2 = data['x\_2'].mean()

* **Что это?** Вычисление средних значений для использования в условии фильтрации.
* **Что делает?** Сохраняет средние арифметические столбцов 'x\_1' и 'x\_2' в переменные mean\_x1 и mean\_x2 соответственно.
* **Как работает под капотом?** Как и в предыдущем сегменте, методы .mean() выполняют быстрые векторизованные вычисления над массивами NumPy, лежащими в основе столбцов DataFrame.

python

print(f"- Среднее x\_1: {mean\_x1:.4f}")

print(f"- Среднее x\_2: {mean\_x2:.4f}\n")

* **Что это?** Вывод вычисленных средних значений для проверки.
* **Что делает?** Показывает, какие пороговые значения будут использоваться для фильтрации. Символ \n добавляет пустую строку для лучшей читаемости.

python

filtered\_data = data[(data['x\_1'] < mean\_x1) | (data['x\_2'] < mean\_x2)]

* **Что это?** **Ключевая строка** этого сегмента — фильтрация DataFrame по сложному условию.
* **Что делает?** Создает новый DataFrame filtered\_data, который содержит только те строки из исходного data, которые удовлетворяют условию: x\_1 меньше своего среднего **ИЛИ** x\_2 меньше своего среднего.
* **Как работает под капотом?** Это мощный механизм Pandas под названием **boolean indexing** (булево индексирование). Процесс можно разбить на шаги:
  1. data['x\_1'] < mean\_x1: Создаётся булев массив (массив из True/False) той же длины, что и DataFrame. На каждой позиции True стоит, если значение в столбце 'x\_1' для этой строки меньше mean\_x1, и False — в противном случае.
  2. data['x\_2'] < mean\_x2: Аналогично создаётся второй булев массив для условия по столбцу 'x\_2'.
  3. |**(Побитовое ИЛИ)**: Два булевых массива объединяются с помощью оператора |. В результирующем массиве True будет стоять там, где True есть хотя бы в одном из исходных массивов. **Важно:** Это именно побитовый оператор |, а не ключевое слово or, потому что мы работаем с массивами, а не с одиночными значениями.
  4. data[...]: Получившийся итоговый булев массив передаётся в качестве индекса в исходный DataFrame data. Pandas выбирает и возвращает только те строки, для которых в массиве стоит True.

python

filtered\_data.to\_csv('lab2\_filtered\_data.csv', index=False)

* **Что это?** Сохранение отфильтрованного DataFrame в новый CSV-файл.
* **Что делает?** Записывает данные из filtered\_data в файл 'lab2\_filtered\_data.csv'. Параметр index=False, как и раньше, гарантирует, что индекс строк не попадёт в файл.
* **Как работает под капотом?** Pandas проходит по всем строкам отфильтрованной таблицы и сериализует их в текстовый формат CSV.

python

print(f"- Отфильтрованный файл 'lab2\_filtered\_data.csv' создан!")

print(f"- Исходных строк: {len(data)}")

print(f"- Отфильтрованных строк: {len(filtered\_data)}\n")

* **Что это?** Финальный отчёт о проделанной операции.
* **Что делает?**
  + Сообщает об успешном создании файла.
  + Сравнивает размер исходного и отфильтрованного наборов данных.
* **Как работает под капотом?**
  + len(data) возвращает количество строк в исходном DataFrame (500).
  + len(filtered\_data) возвращает количество строк в новом, отфильтрованном DataFrame. Это число будет меньше 500, так как условие ИЛИ отбирает довольно большую часть данных (примерно 75%, если данные распределены равномерно).

**Итог по сегменту 5:** Мы успешно применили булево индексирование для фильтрации данных по сложному условию, использующему логическое ИЛИ. Это демонстрирует мощь и выразительность Pandas для операций выборки данных. Результат был сохранён в новый файл, и мы получили количественную оценку эффективности фильтрации.

# Сегмент 6: Построение 3D-графика

python

print("5. Построение 3D-графика.")

* **Что это?** Заголовок для пятого и заключительного этапа работы.
* **Что делает?** Объявляет о начале построения 3D-визуализации.

python

fig = plt.figure(figsize=(12, 8))

* **Что это?** Создание новой фигуры с явным присваиванием переменной.
* **Что делает?** Создаёт объект фигуры размером 12x8 дюймов. В отличие от предыдущего раза, мы сохраняем ссылку на фигуру в переменную fig, чтобы иметь возможность управлять ею напрямую (например, для добавления colorbar).
* **Как работает под капотом?** Создаётся новый объект Figure, который является корневым контейнером для всех элементов визуализации.

python

ax = plt.axes(projection='3d')

* **Что это?** Создание трёхмерных осей на фигуре.
* **Что делает?** Создаёт и возвращает объект осей (Axes3D) с трёхмерной проекцией и присваивает его переменной ax. Теперь все операции с графиком будут выполняться через этот объект.
* **Как работает под капотом?** Модуль mplot3d (который мы импортировали в сегменте 1) регистрирует возможность создания осей с проекцией '3d'. При вызове этой функции создаётся экземпляр класса Axes3D, который "обманывает" matplotlib, заставляя её работать с 3D-координатами.

python

grid\_resolution = 25

* **Что это?** Задание разрешения сетки для 3D-поверхности.
* **Что делает?** Определяет, сколько точек будет использовано по каждой оси для построения поверхности. Значение 25 означает создание сетки 25×25 точек (всего 625 точек). Меньшее значение ускоряет построение, но даёт более грубую поверхность.

python

X1, X2 = np.meshgrid(np.linspace(-2\*np.pi, 2\*np.pi, grid\_resolution), np.linspace(-np.pi, np.pi, grid\_resolution))

* **Что это?** **Ключевая операция** — создание координатной сетки.
* **Что делает?** Генерирует две матрицы X1 и X2 размером grid\_resolution × grid\_resolution, которые представляют все возможные комбинации координат x₁ и x₂ в заданных диапазонах.
* **Как работает под капотом?**
  1. np.linspace(...) создаёт два одномерных массива координат.
  2. np.meshgrid() принимает эти массивы и возвращает две матрицы:
     + X1: Матрица, где каждая строка является копией массива x₁.
     + X2: Матрица, где каждый столбец является копией массива x₂.
  3. В результате, для любой позиции (i, j) в этих матрицах, пара (X1[i, j], X2[i, j]) даёт уникальную координату на плоскости.

python

Y = np.cos(X1 + X2)

* **Что это?** Вычисление значений функции на созданной сетке.
* **Что делает?** Вычисляет косинус от поэлементной суммы матриц X1 и X2. Результат — матрица Y тех же размеров, что и X1, X2, содержащая значения функции в каждой точке сетки.
* **Как работает под капотом?** Векторизованная операция NumPy, применяющая функцию cos к каждому элементу матрицы-суммы.

python

surf = ax.plot\_surface(X1, X2, Y, cmap='viridis', alpha=0.8, linewidth=0, edgecolor='none', antialiased=True)

* **Что это?** Построение 3D-поверхности.
* **Что делает?** Создаёт непрерывную цветную поверхность на основе данных сетки.
  + cmap='viridis': Применяет цветовую карту "viridis" (от фиолетового к жёлтому) для отображения значений Y.
  + alpha=0.8: Задаёт небольшую прозрачность, чтобы видеть задние грани поверхности.
  + linewidth=0, edgecolor='none': Убирает рёбра между полигонами, создавая гладкую поверхность.
  + antialiased=True: Включает сглаживание для улучшения визуального качества.
* **Как работает под капотом?** Matplotlib разбивает поверхность на множество маленьких четырёхугольных полигонов (квадов), каждый из которых заливается цветом в соответствии с его высотой (Y) и выбранной цветовой картой.

python

ax.scatter(X1.flatten(), X2.flatten(), Y.flatten(), color='red', s=1, alpha=1.0, edgecolors='darkred', linewidth=0.5)

* **Что это?** Добавление точечной проекции данных на график.
* **Что делает?** Отображает отдельные точки поверх 3D-поверхности для лучшего восприятия глубины и структуры.
  + .flatten(): "Разворачивает" двумерные матрицы X1, X2, Y в одномерные массивы, так как scatter ожидает плоские списки координат.
  + color='red', s=1: Задаёт красный цвет точек размером 1 пиксель.
* **Как работает под капотом?** Каждая точка отрисовывается отдельно, создавая эффект "песка" на поверхности, что помогает лучше видеть трёхмерную структуру.

python

fig.colorbar(surf, ax=ax, shrink=0.5, aspect=5, label='y')

* **Что это?** Добавление цветовой шкалы.
* **Что делает?** Создает colorbar (легенду), которая показывает соответствие между цветом на поверхности и значением Y.
  + surf: Указывает, для какого объекта строится colorbar.
  + ax=ax: Связывает colorbar с осями ax.
  + shrink=0.5: Уменьшает высоту colorbar до 50% от высоты осей.
  + aspect=5: Задаёт соотношение сторон (ширина/высота) colorbar.
  + label='y': Добавляет подпись к colorbar.
* **Как работает под капотом?** Создаётся новый объект Colorbar, который отображает градиент выбранной цветовой карты и числовые метки, соответствующие диапазону данных.

python

ax.set\_xlabel('x\_1')

ax.set\_ylabel('x\_2')

ax.set\_zlabel('y')

ax.set\_title('3D-график: y = cos(x\_1 + x\_2)')

* **Что это?** Настройка подписей осей и заголовка.
* **Что делает?** Добавляет метки к осям X, Y, Z и заголовок ко всему 3D-графику.
* **Как работает под капотом?** Методы объекта Axes3D устанавливают соответствующие текстовые элементы.

python

plt.tight\_layout()

plt.show()

* **Что это?** Финальная компоновка и отображение графика.
* **Что делает?** tight\_layout() автоматически подстраивает расположение элементов, чтобы они не выходили за границы и не перекрывались. show() отображает финальный результат.

**Итог по сегменту 6:** Мы создали полноценную 3D-визуализацию функции двух переменных, используя комбинацию непрерывной поверхности и точечной проекции. Ключевые концепции — **создание координатной сетки (**meshgrid**)**, **работа с 3D-осями (**Axes3D**)**, **построение поверхностей (**plot\_surface**)** и **добавление вспомогательных элементов (colorbar)**