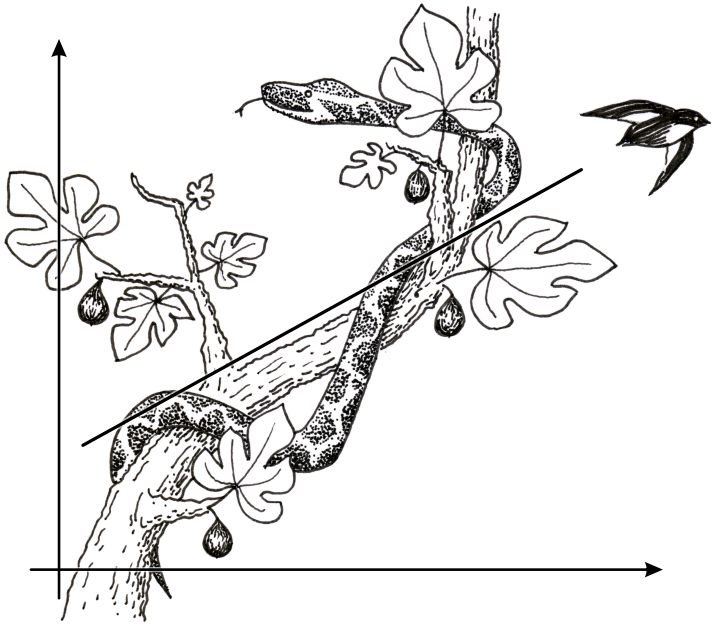
А. Г. Сыромятников, А. Л. Клавсюк

Применение Python для построения графиков в общем физическом практикуме



Москва
Физический факультет МГУ им. М. В. Ломоносова
2026

А. Г. Сыромятников, А. Л. Клавсюк

Применение Python для построения графиков в общем физическом практикуме

Москва
Физический факультет МГУ им. М. В. Ломоносова
2026

А. Г. Сыромятников, А. Л. Клавсюк. **Применение Python для построения графиков в общем физическом практикуме** / Учебно-методическое пособие для студентов младших курсов. — М.: Физический факультет МГУ, 2026. 55 с.
ISBN 978-5-8279-0368-0

В учебно-методическом пособии “Применение Python для построения графиков в общем физическом практикуме” на конкретных примерах задач общего физического практикума рассмотрены основные возможности языка Python и прикладной библиотеки Matplotlib по созданию графиков. Описаны базовые возможности библиотеки Numpy. Разобраны возможности прикладной библиотеки Scipy по регрессионному анализу экспериментальных данных. Даны методические рекомендации по построению графиков. Дан список литературы, который позволит изучить предмет более глубоко, в зависимости от степени первоначальной подготовки. В конце пособия включена краткая справка по описываемому материалу.

Изложение материала не предполагает наличия у читателя предварительного опыта программирования на языке Python.

Учебное пособие предназначено для студентов первого и второго курса физического факультета МГУ имени М. В. Ломоносова, работающих в общем физическом практикуме, но также может быть полезно и студентам старших курсов, аспирантам и сотрудникам.

Рекомендовано Учёным советом Физического факультета МГУ в качестве учебно-методического пособия.

Рецензенты:
к.ф.-м.н., доцент *А. А. Коновко*
д.ф.-м.н., профессор *Д. В. Лукьяненко*

ISBN 978-5-8279-0368-0

Оглавление

1 Введение **4**
- 1.1 Для кого это учебное пособие . . . 4
- 1.2 Принятые условные обозначения . . . 5
- 1.3 Python. Очень краткое введение в язык и прикладные библиотеки . . . 5
  - 1.3.1 Установка программ, настройка окружения . . . 5
  - 1.3.2 Язык программирования Python . . . 7
  - 1.3.3 Среда разработки Jupyter . . . 11
  - 1.3.4 Библиотека Numpy . . . 11
  - 1.3.5 Библиотека Matplotlib . . . 13
  - 1.3.6 Библиотека Scipy . . . 13

2 Требования к графикам **14**

3 Графика в Matplotlib **18**
- 3.1 Составные части рисунка в Matplotlib . . . 18
- 3.2 Представление данных . . . 20
- 3.3 Простейший график. Два подхода к созданию графики в Matplotlib . . . 21
- 3.4 Простой график с погрешностями . . . 22
- 3.5 Несколько графиков на одном рисунке . . . 25
- 3.6 Несколько графиков на разных рисунках . . . 29
- 3.7 Линейная регрессия и построение графика . . . 33
  - 3.7.1 Случай пропорциональной зависимости . . . 34
  - 3.7.2 Случай линейной зависимости . . . 39
- 3.8 Нелинейная регрессия и построение графика . . . 42
- 3.9 График в полярной системе координат . . . 44

4 Дальнейшее изучение **46**

Приложения **48**
- А Краткая справка . . . 48
- Б Листинги примеров . . . 50
- В Описание стилевого файла `ff.mplstyle` . . . 52

Литература **53**

Глава 1. Введение

Известно, что культура оформления результатов измерений разнится от студента к студенту, а требования на Физическом факультете МГУ одинаковые для всех. Во время работы в общем физическом практикуме при обработке результатов эксперимента часто необходимо уметь представить свои результаты графически, в виде графика. При этом это необходимо сделать не абы как, а в соответствии с четкими требованиями.

Это пособие родилось благодаря вихрю однотипных ошибок и вопросов, полученных от студентов во время работы в общем физическом практикуме. Надеемся, что это пособие станет отличным стартом на пути как обработки результатов общего физического практикума, так и в дальнейшей научной жизни.

Разнообразных инструментов для построения графиков имеется огромное количество. Их можно делить по разнообразным категориям, и, очевидно, на вкус и цвет товарища нет. Тем не менее стоит привести основные аргументы, по которым выбор пал именно на Python и Matplotlib.

1. Нулевая стоимость лицензии. Python является свободным программным обеспечением, доступным каждому, у кого есть компьютер.
2. Простота изучения. Язык весьма лаконичен, благодаря чему легко откладывается в памяти.
3. Python изучается на Физическом факультете.
4. Гибкость. Matplotlib предоставляет несколько десятков видов графиков. Кроме того, при необходимости на графике можно изобразить буквально всё что угодно, включая аннотации, растровые изображения и т. п.
5. Востребованность знания языка. Даже если при обучении на Физическом факультете язык не пригодится (в чем, правда, есть некоторые сомнения), есть неплохой шанс, что Python пригодится в дальнейшей жизни.

1.1 Для кого это учебное пособие

Это учебное пособие задумывалось как краткое пособие для студентов первого курса, практически сборник готовых рецептов. При этом предполагается наличие у читателя некоторых минимальных навыков программирования.

[1] https://xkcd.ru/353/

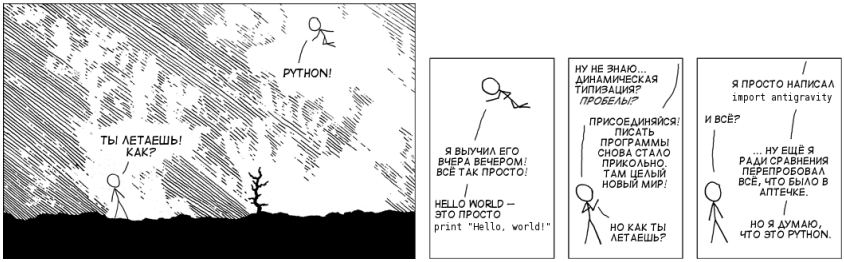


Рис. 1.1: Комикс Рэндела Манро, также известного как xkcd, посвященный Python[1].

1.2 Принятые условные обозначения

В тексте мы будем приводить примеры кода (листинги) на языке Python. Они будут выглядеть следующим образом:

```
1 import this
```

Слева, мелким шрифтом — номера строк. Символ ↪ означает перевод строки. Кстати, если этот листинг на самом деле исполнить в интерпретаторе языка, вы получите короткий текст, позволяющий лучше понять его философию.

Короткие замечания будут выглядеть так.

А в конце книги в приложении Б можно найти полный список всех приведенных примеров.

1.3 Python. Очень краткое введение в язык и прикладные библиотеки

1.3.1 Установка программ, настройка окружения

В качестве рабочего окружения среди всего многообразия вариантов Python будем использовать Anaconda — комплект из пакетного менеджера conda и большого количества предустановленных библиотек, хорошо зарекомендовавший себя для практического использования. Кроме того, он распространяется бесплатно.

Его можно скачать можно на странице загрузки[2], выбрав дистри-

[2] https://www.anaconda.com/download/success

бутив, подходящий для той операционной системы, которая у вас установлена, после чего необходимо его установить. Anaconda уже содержит все необходимые пакеты “из коробки”.

Что касается самого написания кода с целью обработки данных и создания графиков, здесь есть несколько путей.

Первый не требует установки дополнительных программ и предполагает использование среды JupyterLab, работающей в браузере. После установки Anaconda необходимо найти в меню Пуск (или его аналоге) Anaconda Navigator и его запустить. Первый запуск будет весьма длительным, но спустя некоторое время вы увидите прямоугольник с названием JupyterLab и голубой кнопкой Launch (пока не закончится первоначальная инициализация, Navigator будет считать, что JupyterLab не установлен и будет показывать зеленую кнопку Install). После её нажатия запустится веб-сервер JupyterLab и в окне браузера появится приглашение создать свой первый блокнот. Выбрав `Python 3 (ipykernel)`, вы увидите перед собой ваш первый блокнот.

Второй путь предполагает установку одной из сред разработки, например VS Code[3] (бесплатная многоязыковая среда, после установки необходимо будет добавить поддержку Jupyter во встроенном менеджере дополнений) или PyCharm[4] (интегрированная среда разработки на языке Python, платная лицензия, бесплатная студенческая лицензия) или Spyder IDE[5] (бесплатная среда разработки на языке Python, есть поддержка блокнотов через дополнение).

Важно упомянуть, что блокнот — это последовательность кусочков кода, так называемых ячеек (cells). Содержимое каждой ячейки может быть исполнено комбинацией клавиш Ctrl+Enter или кнопкой в панели меню “▶ Run this cell”. При этом результат запуска конкретной ячейки (все создаваемые в ней переменные и их значения, а также измененные в этой ячейке значения переменных, определенных в других ячейках) останется существовать в памяти компьютера либо до её перезапуска, либо до изменения этого результата (или его части) из другой ячейки, либо до конца работы с блокнотом. Это позволяет работать над задачей, разбивая ее на кусочки (например, инициализацию окружения, ввод данных, расчет необходимых величин, отрисовку графика) и постепенно двигаясь дальше с тем, как предыдущие части задачи выполнены.

[3] https://code.visualstudio.com/
[4] https://www.jetbrains.com/academy/student-pack/
[5] https://docs.spyder-ide.org/current/index.html

JupyterLab можно русифицировать установкой соответствующего пакета[a].

[a] https://pypi.org/project/jupyterlab-language-pack-ru-RU/

Опытные пользователи могут не использовать среду разработки, ограничившись текстовым редактором и отдельно установленными интерпретатором языка и библиотеками.

Существует альтернативный способ установки Python, требующий базового умения работы с командной строкой (терминалом).

Вместо Anaconda можно выбрать Miniconda — вариант с существенно меньшим количеством предустановленных пакетов. Он займет гораздо меньше места на жестком диске, а отсутствующие пакеты легко добавить одной командой `conda install имя-пакета`.

Чтобы начать работу с этим пособием, установив Miniconda, необходимо будет сделать следующее:

1. Скачать файл `environment.yml` на жесткий диск. Его можно найти в приложении Б.
2. Открыть командную строку и перейти к расположению этого файла.
3. Выполнить команду `conda env create -f environment.yml` и дождаться окончания установки.

Так будет создано виртуальное окружение `lookatthisgraph`, в котором будут все необходимые нам сейчас пакеты.

Более опытным пользователям этот вариант вполне разумно может показаться более привлекательным. Активацию окружения и запуск JupyterLab в таком случае можно также произвести из командной строки командой `conda activate lookatthisgraph && jupyter lab` без использования Anaconda Navigator.

Теперь вы готовы к работе!

1.3.2 Язык программирования Python

Python — это современный язык программирования, созданный Гвидо ван Россумом (Guido van Rossum) в 1990-е годы (и получивший название в честь знаменитой комедийной труппы “Монти Пайтон”). Хотя Python нельзя назвать идеальным кандидатом для каждого приложения, благодаря своим сильным сторонам он хорошо подходит для многих ситуаций [1].

Ключевыми для нас особенностями языка будут:

1. Общая высокая скорость разработки.
2. Небольшой словарь ключевых слов.
3. Отсутствие необходимости управления памятью. Хотя привычка экономить память и является очень полезной и, в общем, необходимой к приобретению, оставим усилия по ее наработке читателю в качестве домашнего задания.
4. Мультиплатформенность. Один и тот же код будет работать на компьютерах под управлением Windows, Mac OS, Linux и даже Android, при условии, что все необходимые библиотеки установлены.
5. Богатая экосистема прикладных пакетов. Многие частые действия, например, загрузку файлов, рисование графиков, пакетную обработку данных, можно выполнять при помощи сторонних библиотек. Кроме того, зачастую они еще и ускорены за счет использования компилируемых языков.
6. Многочисленное сообщество пользователей. Как правило, в случае проблемы можно без труда найти ее решение в интернете.

При этом основным недостатком языка в виде низкой скорости выполнения кода мы можем смело пренебречь. Во-первых, многие пакеты ускорены за счет наличия в своем составе скомпилированных динамических модулей, написанные не на языке Python, а, например, на C или Fortran. Во-вторых, для небольших задач подход “быстро написано — медленно работает” оказывается гораздо предпочтительнее в виду того, что тогда задача вообще будет решена, а не отложена на неопределенный срок.

В дальнейшем мы не будем заострять внимание на особенностях синтаксиса языка. Всё уже написано до нас. В качестве литературы можно рекомендовать как сравнительно краткие пособия, написанные на Физическом факультете [2, 3], так и подробные книги [1, 4—6]. Кроме того, в качестве настольного справочника удобно использовать книгу [7]. Отдельно необходимо упомянуть очень подробную документацию (на английском языке) на сайте языка[6]. Литературу, рекомендуемую для дальнейшего изучения вопроса, мы привели в разделе 4.

Перед нами сейчас стоит цель не сделать читателя программистом на Python, а научить пользоваться Matplotlib для решения конкретной задачи. Тем не менее некоторые важные особенности языка всё-таки хотелось бы повторить (или рассказать впервые).

1. Существует две версии языка, Python 2 и Python 3. В 2012 году вторая версия объявлена устаревшей, и, говоря “Python”, мы под-

[6] https://docs.python.org/3/

разумеваем “Python 3”. На момент написания книги актуальная версия — Python 3.14.

2. В конце строк не ставится точка с запятой, а при написании очень важны отступы в начале строк — они выделяют в программе отдельные области. Смешение в разных строках в качестве отступов пробелов и символов табуляции ведет к непредсказуемым результатам, но чаще всего к ошибке при выполнении.
3. Если в строке встречается символ #, то та её часть, что следует за ним, отбрасывается интерпретатором. Эта часть называется комментарием и игнорируется при выполнении программы.
4. В отличие от C переменные не имеют строго заданного типа: в мире Python всё на свете является объектами, хранящими тип данных в самих себе. Это называется динамическая типизация. Сравните это поведение, например, с C или C++: там тип данных к переменной “приколочен гвоздями”. Поэтому обязанность следить за типами данных возлагается на программиста. Хотя сразу это кажется неудобством, при получении опыта это становится преимуществом.
5. Чтобы вывести на экран содержимое переменной `X`, достаточно написать `print(X)`. Мы будем выделять это в листингах, добавляя префикс >>>, как это делает интерпретатор.
6. В Python существует богатый набор встроенных типов данных: числа (целые и с плавающей точкой), строки, списки, словари, кортежи, множества (эти четыре типа вместе называют *коллекциями*), файлы. Простейшим примером коллекции в C/C++ является массив. Нас будут интересовать только три типа данных: целые `int` и с плавающей точкой `float` числа (в дальнейшем просто числа), списки `list` и строки `str`. `int` и `float` отличить очень легко: у вторых всегда есть точка, а у первых — нет. Списками в Python называются привычные нам массивы, и определяются они как перечисление своего содержимого, заключенное в квадратные скобки, например `[1, 2.1, 3.2, 4.4]`. Отметим, что в Python нет необходимости хранить в одном списке значения одного типа, хотя поступать мы будем именно так. Мы будем использовать списки для хранения значений некоторой величины. Более подробно о различиях списков и кортежей можно прочесть, например, в книге [3], гл. 5 и 11. Строки же объявляются как свое содержимое, заключенное внутри пары одинаковых одинарных или двойных кавычек, например `'Python'`. Более подробно, например, [3], гл. 3 и 7.

> Часто числа с плавающей запятой `float` неверно называют вещественными числами.

7. Отдельно отметим, что определять тип переменной при ее объявлении не нужно.
8. Python — объектно-ориентированный язык программирования. Поэтому, так как всё — это объект, то у каждой переменной могут быть ассоциированные с ней другие переменные (они называются атрибуты) или функции (их называют методами). Их можно использовать через оператор “точка” (`.`). Так, например, в разделе 1.3.4 чтобы использовать функцию, рассчитывающую синус, реализованную внутри библиотеки Numpy, мы пишем `np.sin()`. Подробнее о работе с модулями и объектно-ориентированном программировании в Python можно прочитать, например, в [3], гл. 3.
9. Ближе к концу книги мы несколько раз воспользуемся таким трюком, как распаковка значений. Суть его в следующем: если в некотором выражении правая часть (от знака равенства) является коллекцией длины N, то левая часть может состоять из N переменных, перечисленных через запятую. В этом случае переменной на месте k будет присвоено значение, соответствующее элементу из этой коллекции с порядковым номером k. При этом вместо явной коллекции справа может стоять функция, возвращающая сразу несколько значений, и этот трюк тоже сработает. Проиллюстрируем примером:

```
a, b, c = [1, 2, 3]
print(c, b)
>>> 3 2     # вывод на экран
```

Для того, чтобы использовать установленные библиотеки, их необходимо импортировать. С этим в языке Python успешно справляется ключевое слово `import`[7]. Второе ключевое слово `as` позволяет дать переменной, ссылающейся на импортированный модуль, любое имя. Да и вообще дословный перевод этой строки звучит как “импортируй что-то как что-то”. Сделаем это:

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

[7] https://docs.python.org/3/reference/simple_stmts.html#import

Теперь мы можем использовать библиотеку Numpy через переменную `np`, а модуль pyplot библиотеки Matplotlib — через `plt`. Такой способ импорта этих библиотек является общепризнанным стандартом. В дальнейших примерах мы будем считать, что библиотеки уже импортированы, а эти две переменные — определены.

1.3.3 Среда разработки Jupyter

Jupyter (https://jupyter.org/) — это выполняемая в веб-браузере среда разработки, использующая вычислимые документы, так называемые блокноты (notebooks), файлы с расширением `.ipynb`. Это гибкая среда, и за счет дополнений она может быть очень тонко настроена. В сообществе, например, Python-программистов формат блокнотов очень широко распространен (пожалуй, за исключением области больших данных и высоконагруженных приложений) и стал де-факто стандартом за счет простоты и удобства работы с ними: все данные, включая неинтерактивную графику (а иногда и интерактивную), сохраняются в файл, что упрощает как хранение, так и возможность поделиться. Также можно использовать Jupyter и с другими языками программирования.

Сравнительно недавно Jupyter эволюционировал в JupyterLab, практически полноценную среду разработки в браузере.

1.3.4 Библиотека Numpy

Numpy (https://numpy.org/) — это фундаментальная библиотека для научных вычислений с Python. Она предоставляет интерфейс для работы с многомерными массивами (`ndarray`, N-dimensional array), а также с многими другими производными объектами (включая массивы с масками и матрицы), а также большое число разнообразных функций, осуществляющих быстрые операции над ними: математические, логические, манипулирующие формой массивов, осуществляющие сортировку, выбор, ввод/вывод, различные преобразования, базовую линейную алгебру, базовые статистические операции, генерацию случайных объектов и многое другое.

Numpy очень хорошо оптимизирована: она полагается в числе прочего на такие библиотеки линейной алгебры, как BLAS и LAPACK. В сообществе Numpy считается обязательным к использованию.

Вначале необходимо объяснить некоторые важные моменты работы с массивами, включая транслирование (broadcasting).

Массив Numpy обладает тремя характеристиками: типом данных, размером (его называют формой, `shape`) и собственно, самими данными.

В отличие от Python вообще, массивы Numpy обладают статической

строгой типизацией: невозможно, например, засунуть в массив целочисленных значений одно вещественное. Тип данных в массиве задается при его создании — либо копированием из существующего, либо при создании "с нуля" — ключевым словом `dtype`. Обычно при создании массива из уже существующего списка значений Numpy угадывает тип данных, основываясь на имеющихся значениях.

Вторая отличительная особенность Numpy — транслирование. Оно позволяет совершать операции над массивами разной размерности, производя над ними неявные преобразования. Оно состоит из двух правил:

1. Если не все входные массивы имеют одинаковое число измерений, то к тем, у кого размерность ниже, к их форме спереди добавляется единица, пока размерности всех массивов не станут одинаковыми.
2. Одномерные массивы длиной, равной 1, вдоль некоторой оси ведут себя так, как если бы они имели вдоль этой оси такой же размер, как и другие массивы, входящие в выражение. Каждое значение в массиве вдоль этой оси будет равно тому единственному, которое там было изначально.

Проиллюстрируем эти правила примерами. Обратите внимание, что массивы Numpy необходимо создавать явным образом (см. первые три строки).

```
a = np.array([1, 2, 3])      # массив целых чисел
b = np.array([1., 2., 3.])   # массив вещественных чисел
x = np.array([1, 2, 3], dtype=float)
# массив вещественных чисел с явно заданным типом

print(a.dtype, b.dtype)      # проверим
>>> int32 float64

# Транслирование согласно правилу 1:
# массивы одинаковой формы (длины).
c = a + b    # Эта запись эквивалентна c_i = a_i + b_i для любого i
             # При этом форма c будет такой же, как у a и b.
print(c)
>>> [2. 4. 6.]
# Кроме прочего, случилось приведение к более полному
# типу данных: от целочисленного к вещественному.
```

```
15 # Транслирование согласно правилу 2: 3 - это скаляр.
16 d = 3 * a   # Эта запись эквивалентна d_i = 3 · a_i для любого i
17 print(d)
18 >>> [3 6 9]

19 # Транслирование можно использовать и с методами Numpy!
20 e = np.sin(a)
21 # Эта запись эквивалентна e_i = sin(a_i) для любого i
22 print(e)
23 >>> [0.84147098 0.90929743 0.14112001]
```

Больше информации можно найти на странице "Быстрый старт" документации к Numpy[8].

1.3.5 Библиотека Matplotlib

Matplotlib (https://matplotlib.org/) — библиотека на языке программирования Python для визуализации данных при помощи двумерной и трёхмерной графики.

Возможности Matplotlib очень велики: с его помощью можно создавать несколько десятков различных видов графики. Пользователь может указать оси координат, сетку, добавить надписи и пояснения, использовать логарифмическую шкалу или полярные координаты. При этом каждый элемент при желании можно очень тонко настроить. Получившиеся графики можно сохранить в один из многих поддерживаемых форматов, включая pdf, png, svg, jpg и т.д.

Кроме прочего, есть очень удобная шпаргалка[9] по использованию библиотеки.

1.3.6 Библиотека Scipy

Scipy (https://scipy.org/) — высокооптимизированная библиотека, предоставляющая доступ к различным алгоритмам оптимизации, интегрирования, интерполяции, дифференцирования, работы со статистикой, а также решения алгебраических и дифференциальных уравнений, задач на собственные значения и многих других классов задач. Это в основном широко применимые алгоритмы, которые могут пригодиться в самых разных областях науки. Scipy также расширяет Numpy, предоставляя дополнительные инструменты для работы с многомерными массивами, а также разреженными матрицами и k-мерными деревьями.

[8] https://numpy.org/doc/stable/user/quickstart.html
[9] https://matplotlib.org/cheatsheets/cheatsheets.pdf

Глава 2. Требования к графикам

Опишем основные требования к графическому представлению данных (графикам). В каждом издательстве научных журналов требования к графикам различаются. Поэтому необходимым навыком ученого является еще и умение оформить свой труд в соответствии с предъявляемыми требованиями. Список требований в общем физическом практикуме определен в пособии [8]. Пройдемся по его пунктам и выясним, что выполняется автоматически, а что необходимо настроить отдельно. Правила, которые не нарушаются, выделены звездочкой ✱. Правила, выполняемые автоматически, отмечены значком ✔ (если их, конечно, не сломать намеренно).

При этом важно помнить, что основным правилом для вас должен быть здравый смысл.

Значком 📊 мы отметим те пункты, которые можно вместо явных команд настроить специальным стилевым файлом в формате `matplotlibrc`. Это обычный текстовый файл, его формат и возможные опции подробно описаны в документации[1]. Стилевой файл, названный `ff.mplstyle`, можно найти в приложении Б, а его подробное описание — в приложении В. После его подключения все эти требования будут вами выполнены.

1. ✱ *Независимая переменная x откладывается по оси абсцисс, а функциональная зависимость $y = f(x)$ — по оси ординат.*
   В Matplotlib независимая переменная идет в списке аргументов всегда перед зависимой, так что это — вопрос внимательности. Опуская необязательные аргументы, вызов любой функции построения графиков выглядит как `plt.plot(x, y, ...)`.
2. 📊 *Для построения графиков используются координатные оси или сетка. При построении графиков на компьютере координатная сетка обязательна.*
   Matplotlib автоматически создает прямоугольную область, ограниченную краями осей (spines) для каждых осей. Для добавления собственно сетки необходимо в область настройки рисунка добавить явную просьбу её нарисовать `plt.grid(True)` (или `ax.grid(True)` при явном подходе.). Выключить сетку можно, очевидно, командой `plt.grid(False)`.

[1] https://matplotlib.org/stable/users/explain/customizing.html

> Обратим, однако, внимание, что пользоваться координатной сеткой надо с умом: она не должна отвлекать от представляемых данных. Поэтому во многих научных журналах координатная сетка, наоборот, нежелательна.

3. 📊 *Координатные штрихи направлены в область, где будет построен график, с другой стороны будут нанесены координатные числа.*
   По умолчанию в Matplotlib деления на осях смотрят наружу. Это можно изменить, настроив отображение делений функцией `plt.tick_params(axis='both', which='both', direction='in')` (или `ax.tick_params(...)`).
4. *Оси могут начинаться не с нуля. Если обе оси начинаются с нуля, то "0" ставится только один раз.*
5. ✔ *Данные должны располагаться на поле графика так, чтобы не оставалось больших пустых пространств.*
6. ✱✔ *Если на графике есть и положительные, и отрицательные значения, то ось обязательно проходит через "0".*
   Matplotlib сам выбирает диапазон значений на основании отображаемых данных. Настройка пределов этого диапазона и трюка, необходимого, чтобы ноль был лишь один, обсуждаются в разделе 3.5.
7. 📊 *Соблюдайте толщину линий, размеры экспериментальных точек, размер шрифта подписей. График должен быть ярким и контрастным.*
   Хотя Matplotlib самостоятельно делает толщины графиков и размеры точек разумными, их можно настроить при помощи следующих параметров:
   - Толщину линии при помощи параметра `lw` или `linewidth` функции `plt.plot`: `plt.plot(x, y, lw=2, ...)`.
   - Размеры точек при помощи параметра `s` (size) функции `plt.scatter` или при помощи параметра `ms` (marker size) функций `plt.plot` или `plt.errorbar`.
   - Размер шрифта подписей к осям или легенде можно настроить параметром `fontsize` функций `plt.(xy)label`, `plt.title`, `plt.legend`.

   Численные значения параметров подбираются из соображений здравого смысла.
8. ✔ *Лист должен быть максимально заполнен, но всё должно поместиться.*
   Matplotlib сам выбирает разумные размеры для графиков и отдельных элементов. При желании их можно тонко настроить,

однако это выходит за рамки этого пособия.

9. ✱ ✔ *Если масштаб графика равномерный, то координатные штрихи и числа расставляются равномерно.*
10. ✱ ✔ *На осях обычно делают 5–10 делений, рядом с ними наносят их числовые значения. Измеренные значения на шкалы не наносят.*
11. ✱ ✔ *Масштаб выбирают удобным для считывания.*
    Тонкая настройка значений на осях обсуждается в разделе 3.7.1.
12. ✱ *На осях должны быть обозначены изображаемые переменные величины и их единицы измерения.*
    По умолчанию Matplotlib располагает подписи к осям посередине параллельно им. Добавление подписей обсуждается в разделе 3.3. Мы рекомендуем использовать наименование величины и её символ (если есть), а после, через запятую, единицу измерения. Пример: “Напряжение U, мВ” или “Вероятность P” или “Число отсчетов”.
13. 📊 *Погрешности измерения каждой точки указываются отрезками, длина которых равна величине ошибки.*
    Погрешности подробно обсуждаются в разделе 3.4.

Использовать стилевой файл легко. Для этого его необходимо положить в рабочую папку — ту, в которой лежит блокнот, в котором вы работаете. Далее сразу после импортов библиотек используется конструкция

```
1 plt.style.use('ff.mplstyle')
```

Во всех примерах мы используем стилевой файл. Его содержание подробно описано в приложении B.

В качестве справочного материала приведем требования к иллюстрациям редакции журнала “Письма в ЖЭТФ”[2].

Поскольку рисунки переносятся без изменений из “Писем в ЖЭТФ” в “JETP Letters”, все надписи на рисунках должны быть только на английском языке. Авторов, использующих при подготовке рисунков компьютерную графику, просим придерживаться следующих рекомендаций:

- графики делать в рамке;
- штрихи на осях направлять внутрь;
- по возможности использовать шрифт Times;

[2] http://jetpletters.ru/ru/info.shtml

- высота цифр и строчных букв должна быть в пределах (3–4)% от максимального размера (высоты или ширины) рисунков, это относится и к цифрам на осях вставки;
- единицы измерения на осях графиков приводить в скобках.

При подготовке рисунка имейте в виду, что, как правило, ширина рисунка при печати не превышает 82 мм; в исключительных случаях рисунок размещается на всей ширине листа (до 160 мм).

Рисунки публикуются “on-line” в цвете. На авторов возлагается обязанность проверить, что цветные рисунки достаточно контрастны и читаемы в черно-белом печатном варианте.

Глава 3. Графика в Matplotlib

3.1 Составные части рисунка в Matplotlib

Прежде чем мы начнем строить графики, необходимо разобраться, с чем мы будем иметь дело. Давайте выясним, из чего состоит типичный рисунок. Для этого построим рисунок с двумя линиями и набором точек (см. рис. 3.1). Каждый элемент рисунка подписан, а также внизу указаны методы, которые создают тот или иной элемент. В дальнейшем им будет удобно пользоваться как кратким руководством. Далее будем использовать терминологию, введенную в этой главе.

- Рисунок (Figure) — это, собственно, сам рисунок. Он включает в себя координатные оси, данные, подписи и т. д. Он создается при вызове (явном или неявном) метода `pyplot.figure`. В этот момент проще всего задать фон рисунка, его размер и многие другие параметры.
- Оси (Axes) — область рисунка, на которой будут отображаться данные, ограниченная по краям координатными осями. Это та область рисунка, на которой могут располагаться данные. На одном рисунке может быть несколько осей, они могут быть упорядочены разными способами (см. главу 3.6).
- Координатная ось (Axis) — это линия, на которую нанесены деления, как большие, так и малые. Рядом с каждой из осей может быть ее название (label).
- Заплатки (Patches) — различные двумерные части рисунка.
- Линии (Lines) — линии графиков.
- Маркеры (Markers) — точки точечных графиков.
- Легенда (Legend) — подпись с обозначениями, какой график какие данные отображает.

Элементы рисунка сортируются по виртуальной глубине `zorder` (порядок вдоль оси z): грубо говоря, что будет отрисовываться на чём. Наглядно этот порядок представлен на рисунке 3.2. Чем больше `zorder`, тем выше слой в виртуальной стопке. В общем случае каждый новый отрисованный элемент рисуется поверх предыдущего с таким же `zorder`. Явное изменение `zorder` может использоваться для, например, отрисовки маркеров над линиями (см. пример здесь[2]).

[1] https://matplotlib.org/stable/gallery/showcase/anatomy.html
[2] https://matplotlib.org/stable/gallery/misc/zorder_demo.html

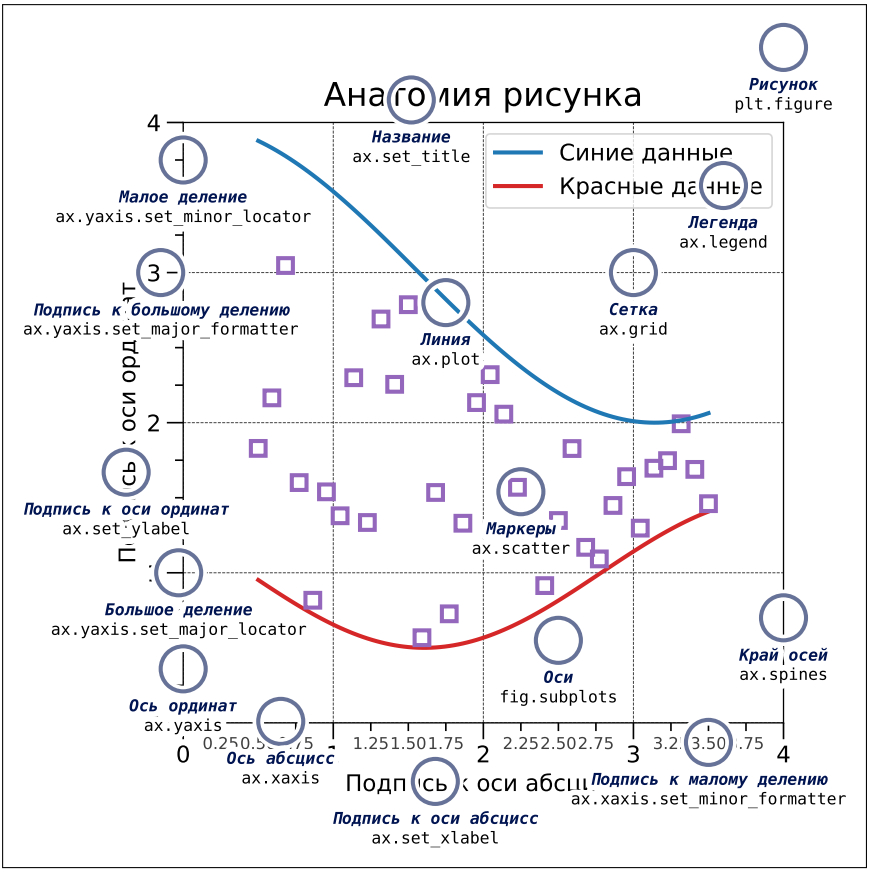


Рис. 3.1: Анатомия рисунка Matplotlib[1]. Под названием каждого элемента указан метод, его создающий.

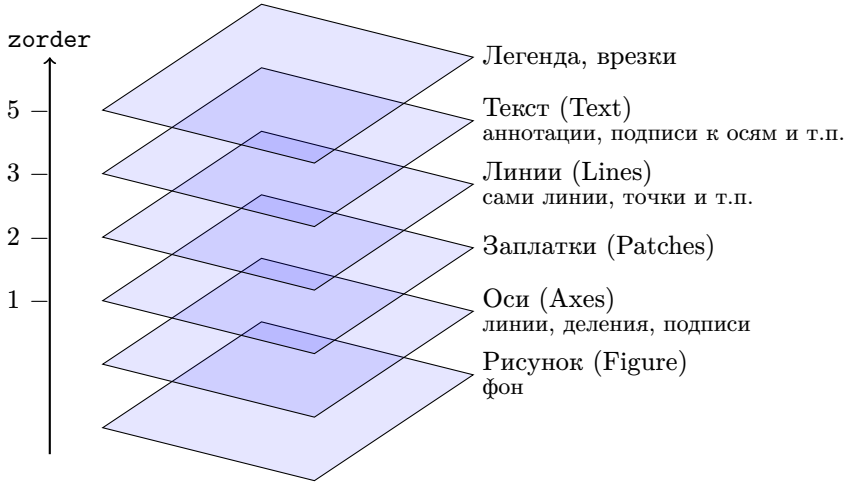


Рис. 3.2: Виртуальная глубина `zorder`. Для некоторых слоев указано значение по умолчанию.

Руководствуясь знанием английского языка, следует отличать слова *axis* (ось, единственное число) и *axes* (оси, множественное число).

3.2 Представление данных

Мы будем представлять данные двумя разными способами, в виде списков и в виде n-мерных массивов Numpy. Первым способом будем пользоваться, когда данные не предполагаются к дальнейшему изменению перед их отрисовкой. Это более лаконично и, в принципе, делает импорт Numpy необязательным.

```
1 x = [0, 0.2, 0.4, 0.6, 0.8] # c
```

Удобно записывать рядом с исходными данными их размерность.

Второй способ основывается на первом, когда из уже определенного списка мы создаем `ndarray` библиотеки Numpy, что дает существенно бóльшие возможности. Напомним, что массивы Numpy строго типизированы. Тип данных их содержимого (`dtype`) может быть вычислен

по элементам передаваемого списка. В приведенном ниже примере в каждом списке есть как минимум одно вещественное число, поэтому это, скорее всего, будет `float64`, то есть вещественные числа двойной точности, занимающие в памяти компьютера по 64 бита каждое. На некоторых компьютерах разрядность может быть равна 32 бита. Такой точности в подавляющем большинстве случаев хватает с избытком.

```
y = [1.35, 2.68, 4.04, 5.37, 6.71]  # просто список Python
y = np.array(y)                     # стал n-мерным массивом
# или просто
y = np.array([1.35, 2.68, 4.04, 5.37, 6.71])

print(y.dtype)
>>> float64
```

Можно заставить Numpy создать массив определенного типа данных, передав аргумент `dtype`:

```
x = np.array([1, 2, 3], dtype=float)      # [1. 2. 3.]
```

3.3 Простейший график. Два подхода к созданию графики в Matplotlib

Пусть в результате выполнения задачи общего физического практикума была измерена зависимость координаты груза от времени (машина Атвуда, задача 101). Зададим эти данные в виде двух списков, `t` и `x`:

```
t = [1, 2, 3, 4, 5]                          # секунды
x = [0.241, 0.276, 0.3335, 0.414, 0.5175]    # метры
```

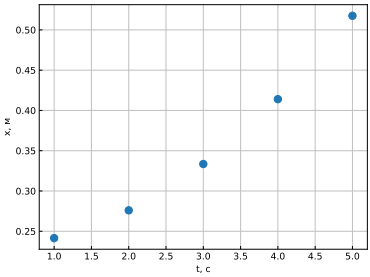
При создании графиков в Matplotlib используются два различных подхода, явный и неявный. В первом из них мы явно создаем рисунок, оси и так далее. Так мы получаем большую гибкость в работе за счет возможности работать с каждым объектом явно. Это приводит к более многословному описанию рисунка, но такой подход всецело оправдывает себя в случае сложной графики. Мы используем его в более сложных примерах, начиная с раздела 3.6.

Неявный подход заключается в том, что любые манипуляции мы будем проводить с переменной `plt`, а возвращаемые при этом значения

нас будут интересовать лишь в особенных случаях. При этом Matplotlib сам неявно создаст все необходимые объекты вроде осей и тому подобного. Построим точечный график зависимости $x(t)$ с использованием неявного подхода:

```
1 import matplotlib.pyplot as plt
2 plt.style.use('ff.mplstyle')

3 plt.scatter(t, x)
4 plt.xlabel("t, с")
5 plt.ylabel("x, м")
6 plt.savefig("example_1_simple.pdf"
```



Разберём этот пример построчно.

1–2. Импорт библиотеки и подключение стилевого файла.

3. Здесь *неявно* был создан рисунок стандартного размера (8 дюймов в ширину и 6 дюймов в высоту, примерно 20×15 см), созданы оси (единственные на рисунке), построены отметки на осях некоторым наилучшим по мнению Matplotlib образом и нарисованы точки размера по умолчанию, соответствующие нашим данным и раскрашенные первым по порядку цветом во внутренней таблице цветов библиотеки — синим.

4–5. Добавление подписей к осям.

6. Эта строка сохраняет рисунок в файл. Формат определяется по переданному функции имени файла, в данном случае это pdf-документ `example_1_simple.pdf`.

Пример целиком приведен в конце книги в разделе Б.

3.4 Простой график с погрешностями

В задаче общего физического практикума про баллистический маятник (задача 103) необходимо построить зависимость средней силы удара пули об маятник от её массы. Расширим предыдущий пример, учтя необходимость построить погрешности вычисленных нами величин силы. В примере список `df` — стандартные отклонения силы удара, рассчитанные нами отдельно.

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

```
3  m =  [0.00301, 0.0064, 0.0102, 0.0129, 0.0161, 0.0255]   # кг
4  f =  [0.029, 0.0303, 0.0309, 0.0319, 0.0328, 0.0344]     # Н
5  df = [0.0022, 0.0046, 0.003, 0.0025, 0.002, 0.0014]      # Н

6  plt.errorbar(m, f, yerr=df, fmt='o', capsize=5)
7  plt.xlabel('m, кг')
8  plt.ylabel('<f>, Н')

9  plt.title('Зависимость средней силы удара от массы пули')
10 plt.savefig('example_2_errors_kg.pdf')
```

Для начала обратим внимание на последовательность команд, рисующих график.

1–2. Подготовка окружения, здесь это импорты библиотек.
3–5. Подготовка данных.
6. Создание рисунка (в данном случае неявное), осей, отрисовка данных.
7–9. Настройка рисунка, добавление подписей и прочего.
10. Сохранение рисунка в файл.

Конечно, такое деление условно, оно продиктовано лишь желанием структурировать программный код, сделать его лёгким для чтения и изменения. Очевидно, команды в строках 3–8 могут повторяться сколько угодно раз, пока требуемый результат не будет достигнут (если, например, осей несколько, как в разделе 3.6).

Разберем строку 6 более подробно. С первыми двумя аргументами мы уже знакомы: это массивы значений по осям x и y, соответственно. Погрешность величины `f` определяется аргументом `yerr=df`. В данном случае это массив такой же длины, как и `f`, а это значит, что он будет понят как симметричная погрешность, одинаковая как сверху, так и снизу.

В строках 7 и 8 происходит добавление подписей к осям. Подписи могут содержать как латинские или русские буквы, так и многие символы из таблицы Юникод. Более того, здесь можно использовать математическую запись MathText, подробно описанную на стр. 26. Несложно догадаться, что подпись к оси абсцисс добавляется функцией `plt.xlabel()`, а к оси ординат — `plt.ylabel()`.

Не забывайте добавлять к тексту размерность изображаемой величины.

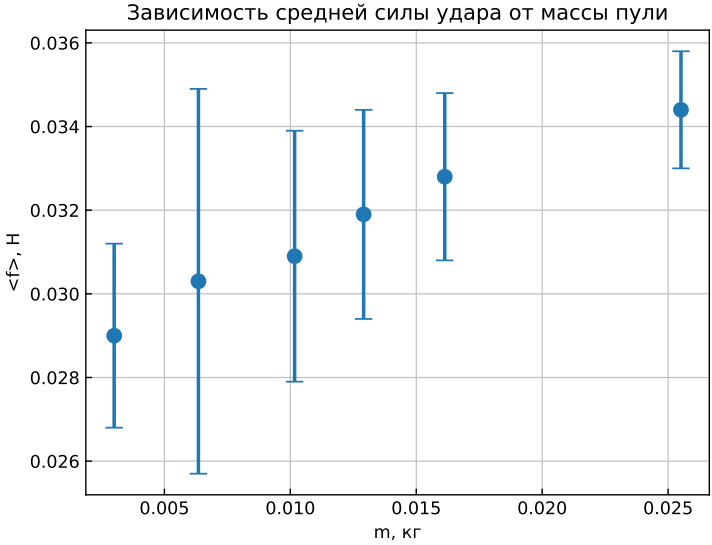


Рис. 3.3: Результат построения графика с погрешностями

Далее следует аргумент `fmt='o'`. Строки форматов в Matplotlib заслуживают отдельного описания. Вообще, они являются комбинацией[3] трех однобуквенных обозначений значка маркера, стиля линии и цвета: `fmt = '[marker][line][color]'`. Но в данном примере ни линия, ни цвет нас не интересуют, поэтому оставляем только обозначение маркера. Несколько других возможных вариантов есть на рисунке ниже. Полный список маркеров можно найти в документации[4].

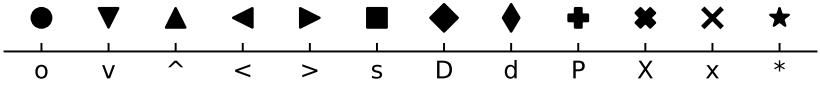


Рис. 3.4: Различные стили значков маркеров и их обозначения

Последний аргумент `capsize=5` определяет размер шляпок на концах отрезков, определяющих погрешности. Впрочем, это уже определено в стилевом файле `ff.mplstyle`.

Наконец при помощи `plt.title` мы добавляем графику подпись, “Зависимость средней силы удара от массы пули”. Такая подпись мо-

[3] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.plot.html
[4] https://matplotlib.org/stable/api/markers_api.html

жет быть только одна, по умолчанию она будет находиться сверху. В результате получится следующий график.

С эстетической точки зрения необходимо убрать с подписей к делениям оси абсцисс излишнее количество нулей и запятых, заменив единицы измерения с килограммов на граммы. Для этого необходимо домножить соответствующую величину на 1000 и поменять подпись к оси. Исправим наш пример в районе 6 строки:

```
6 m = np.array(m)
7 m *= 1e3
8 ...
9 plt.xlabel('m, г')
```

В результате ось абсцисс будет выглядеть существенно лучше.

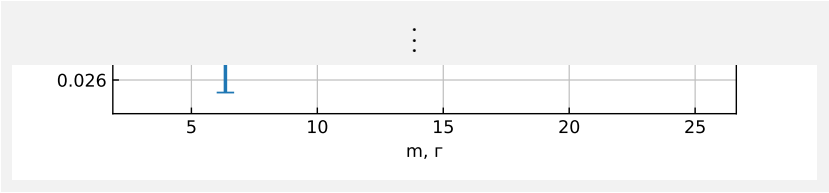


Рис. 3.5: Исправленная ось абсцисс рисунка 3.3

3.5 Несколько графиков на одном рисунке

В задаче 102 необходимо получить зависимости квадрата скорости тележки от пройденного ей расстояния для случаев разной массы тележки и разного угла наклона самой скамьи. Если проводить опыт для трёх разных масс и трёх разных значений наклона, получится $3 \times 3 = 9$ наборов данных. Логично объединить их по некоторому признаку и сократить число графиков. В данном примере мы построим все данные для тележки одной массы. Для краткости опустим настройку окружения и определение массивов данных `x`, `v2_1`, `v2_2` и `v2_3`.

```
1 plt.scatter(x, v2_1, label=r'$\alpha = 0.11^\circ$')
2 plt.scatter(x, v2_2, label=r'$\alpha = 0.23^\circ$')
3 plt.scatter(x, v2_3, label=r'$\alpha = 0.37^\circ$')
4 plt.xlim(xmin=0)
5 plt.ylim(ymin=0)
6 plt.xlabel('$x$, м')
```

```
7  plt.ylabel('$v^2, м^2/с^2$')
8  plt.title('Зависимость квадрата скорости тележки $v^2$ от ... $x$')
9  plt.legend(title='Наклон скамьи')
10 plt.savefig('example_3_sameaxis.pdf')
```

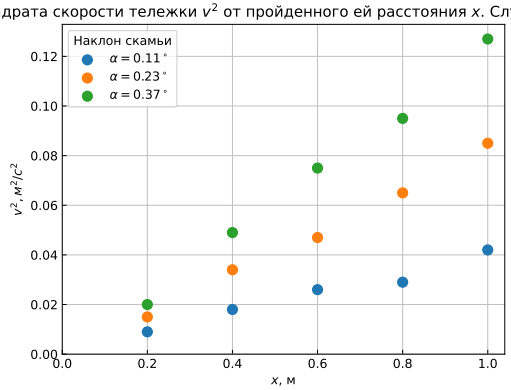


Рис. 3.6: Результат построения нескольких графиков на одном рисунке

Разберем построчно, что здесь происходит.

1–3. Создание рисунка, отрисовка трёх наборов данных.
4–5. Определение пределов для отображаемых данных по осям абсцисс и ординат.
6–8. Добавление подписей к осям и самому рисунку.
9. Добавление легенды к рисунку.
10. Сохранение рисунка в файл.

В первых строках 1–3 в уже известный нам вызов `plt.scatter` добавился новый аргумент `label`. Это строка, которая будет описывать график в легенде к рисунку. Обратим внимание на внешний вид этой строки, а именно на её часть, обрамленную символами `$`. У Matplotlib есть для неё особое название, MathText, и её отрисовка происходит по иным правилам. Текст между `$` разбирается на кусочки, токены, и те из них, что начинаются с обратной косой черты, заменяются на математические символы аналогично тому, как делает система компьютерной вёрстки LaTeX. При этом количество пробелов между отдельными токенами неважно. Некоторые токены могут принимать аргументы, которые определяют их вид. Это, например, радикал $\sqrt{\cdots}$ (`\sqrt{}`), тригонометрические функции, дробь $\frac{\cdots}{\cdots}$ (`\frac{}{}`).

При помощи MathText можно вставить буквы греческого алфавита, под- и надстрочные индексы, дроби и многое другое. Примеры приведены в таблице ниже.

`$\alpha \beta \gamma \epsilon \lambda$`	→	$\alpha\beta\gamma\epsilon\lambda$
`$\mu \nu \pi \rho \sigma \tau \varphi$`	→	$\mu\nu\pi\rho\sigma\tau\varphi$
`$\tan\theta_d + x^2 \times y^\prime$`	→	$\tan\theta_d + x^2 \times y'$
`$\sin\sqrt{4\omega} - \arccos{\delta}$`	→	$\sin\sqrt{4\omega} - \arccos\delta$
`$\frac{1}{\eta}, \frac{кг}{м \cdot c^2}$`	→	$\frac{1}{\eta}, \frac{кг}{м \cdot c^2}$

Обратите внимание на использование именно фигурных скобок `{}`, а также, что начертание MathText курсивное, то есть с небольшим наклоном. Поэтому из эстетических соображений рекомендуется использовать MathText или для всех обозначений, или не использовать вообще. Полный список правил и символов можно найти в документации[5].

Иногда начало токена совпадает с управляющим символом и интерпретируется языком Python иначе, чем мы этого ожидаем. Всего управляющих символов[a] немного: `\0`, `\a`, `\b`, `\e`, `\t`, `\n`, `\v`, `\f` и `\r`. В таком случае надо превратить обычную строку Python в "сырую" (raw), в которой интерпретация управляющих символов не происходит. Это делается путем добавления `r` перед первой кавычкой: `'$\tan\theta$'` → `r'$\tan\theta$'`.

[a] https://docs.python.org/3/reference/lexical_analysis.html#escape-sequences

Хотя подавляющее большинство обозначений токенов для MathText взято из LaTeX, в котором, например, свёрстано это пособие, правила их использования в Matplotlib гораздо проще, за что приходится платить гибкостью. Так, например, подружить дроби и кириллицу в LaTeX несколько сложнее, чем в Matplotlib. Опытный пользователь может включить отрисовку MathText средствами LaTeX, предварительно его установив.

Заметим, что если тележка проедет 0 метров, то ее скорость будет так же равна нулю. Это означает, что на графике должна присутство-

[5] https://matplotlib.org/stable/users/explain/text/mathtext.html

вать точка (0, 0), хотя автоматические пределы на осях её исключили. Для её отображения мы должны изменить пределы отображения данных, поменяв нижний предел обеих осей на 0 в строках 4 и 5. Сделаем это при помощи функций `plt.xlim(xmin=число, xmax=число)` и `plt.ylim(ymin=число, ymax=число)`, учтя, что любой из аргументов можно опустить.

Учитывая, что графиков на рисунке несколько, мы должны их както различать. Ранее мы уже дали каждому из них своё имя, теперь надо отобразить их на рисунке. Сделаем это, создав легенду графика при помощи `plt.legend`. Вызов её без аргументов поместит легенду для всех графиков с объявленной `label` в том месте рисунка, где наименьшее количество точек данных, в данном случае слева сверху. Это можно изменить при помощи аргумента `loc`. Список возможных его значений описан в документации[6]. Аргументом `title` мы добавляем подпись к легенде, и сразу становится ясно, что за величина α.

Несмотря на то, что мы нигде не объявляли цвета графиков, они тем не менее разные. Цвет графика можно задать, передав внутри вызова `plt.scatter` или `plt.errorbar` аргумент `color`. Его значение может относиться к одной из следующих групп (список не полный):

1. Однобуквенное обозначение одного из 8 основных цветов (см. рисунок ниже)
2. Строку с названием цвета, например `'orange'`. Полный список собран здесь[7].
3. Строку вида `'Cx'`, где x — целое число. Тогда цвет будет выбран из набора цветов по умолчанию (см. рисунок ниже). Эти цвета повторяются до бесконечности с периодом, равным 10.

Полный обзор вариантов описан в документации[8].

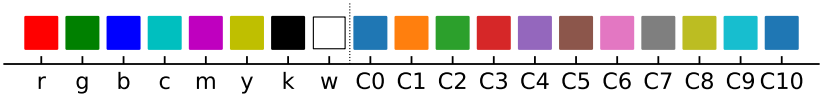


Рис. 3.7: Некоторые возможные цвета элементов графика и соответствующие им значения `color`

[6] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.legend.html
[7] https://matplotlib.org/stable/gallery/color/named_colors.html
[8] https://matplotlib.org/stable/gallery/color/color_demo.html

Иногда при сохранении рисунка в файл бывает, что его часть обрезается или наоборот, остаются широкие белые поля. Это связано с тем, что конечный размер рисунка не был рассчитан при сохранении в файл. Избежать этого можно, добавив в вызов `plt.savefig` явную просьбу его вычислить.

```
plt.savefig('example_3_sameaxis.pdf',
↪   bbox_inches='tight')
```

Сделаем некоторые косметические изменения.

1. Заметим, что в нижнем левом углу графика есть и `0.00`, и `0.0`. От одного из нулей можно избавиться, если изменить пределы по одной из осей до очень маленькой не равной нулю величины. Например, `plt.ylim(ymin=1e-6)` в строке 5.
2. Название графика в нашем случае очень длинное. Разобьем его на несколько строк управляющим символом перевода строки `'\n'` в строке 8: `plt.title('Зависимость... $v^2$ \n от ...')`.

В результате получится следующий график.

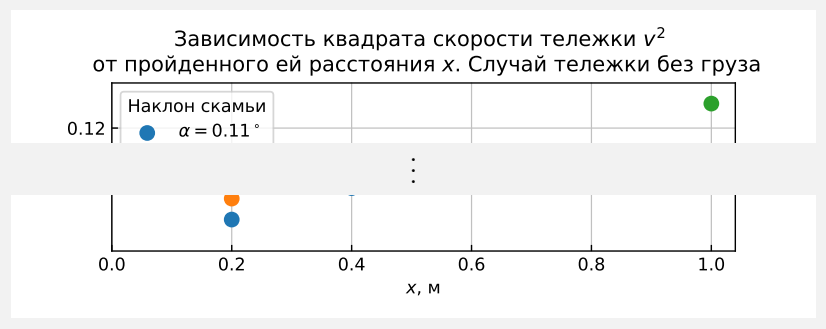


Рис. 3.8: Исправленные заголовок графика и пределы по осям рисунка 3.6

3.6 Несколько графиков на разных рисунках

До текущего момента использовался неявный подход, и для рисунка, состоящего из единственных осей, его было достаточно. Для создания графика с несколькими осями лучше использовать явный подход: будем определять рисунок и оси самостоятельно и работать уже с ними напрямую. Важно, что при этом имена большинства методов изменятся.

В задаче Э4.1 в разделе “Электромагнетизм” общего физического практикума необходимо измерить зависимость силы тока через полупроводниковый диод от напряжения (вольт-амперную характеристику, ВАХ). В задаче требуется построить как саму ВАХ $I(U)$, так и зависимость $\ln\left(I(U)/I_0\right)$, чтобы впоследствии определить на ней линейный участок. При этом важно синхронизировать масштаб по оси абсцисс. Будем считать, что экспериментальные данные мы уже ввели в списки `I` и `U`.

```
1  fig, axes = plt.subplots(2, 1, figsize=(5, 8))
2  ax1, ax2 = axes

3  ax1.scatter(U, I)
4  ax1.set_title('ВАХ полупроводникового диода')
5  ax1.set_xlabel('U, мВ')
6  ax1.set_ylabel('I, мА')
7  ax1.set_xlim(xmin=300, xmax=650)
8  ax1.set_ylim(ymin=0, ymax=1.4)

9  I = np.array(I)
10 ax2.scatter(U, np.log(I))
11 ax2.set_xlabel('U, мВ')
12 ax2.set_ylabel('ln I/I$_0$')
13 ax2.set_xlim(300, 650)
14 ax2.set_ylim(-3, 0.5)

15 fig.savefig('example_4_multipleaxes_1.pdf', bbox_inches='tight')
```

Разберем пример построчно.

1. Явное создание набора из двух пар координатных осей.
2. Распаковка `axes` в отдельные переменные для удобства использования.

3–8. Отрисовка и настройка верхнего графика.

9–14. Отрисовка и настройка нижнего графика.

15. Сохранение готового рисунка в файл.

В строке 1 мы определяем рисунок (figure) шириной 5 и высотой 8 дюймов. Так как график виртуальный, правильнее относиться к этому размеру как к соотношению сторон. Этот рисунок будет состоять из 2 строк и 1 колонки координатных осей (первые два аргумента). Вызов `plt.subplots` возвращает два значения, сам рисунок и список осей.

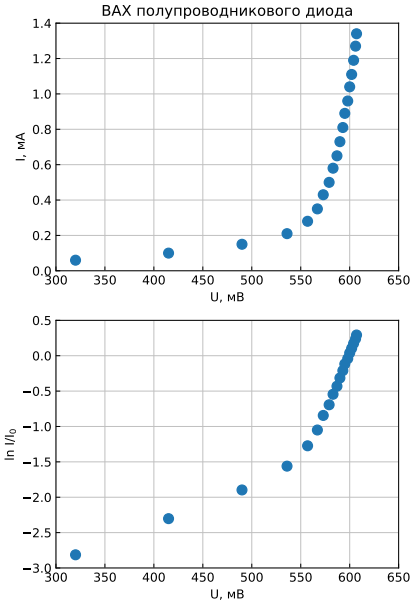


Рис. 3.9: Результат построения двух графиков на разных рисунках

Пользуясь распаковкой (раздел 1.3.2), мы сохраняем эти значения в две разные переменные, `fig` и `axes`. Важно, что массив `axes` будет такой же формы, какой мы задали рисунок при вызове `plt.subplots`. Например, мы создали рисунок 2×1 оси, значит и массив `axes` будет размером 2×1. Это позволит нам удобно распаковать `axes` в переменные `ax1` и `ax2`. Еще несколько примеров есть в документации[9].

> Обратим внимание на то, что от размера рисунка зависит относительный размер шрифтов: на рисунке размером 2×3 дюйма буквы будут крупнее, чем на рисунке 6×9 дюймов.

Описание графиков дальше разделено на два больших блока, заканчивающихся сохранением рисунка в файл (обратите внимание, что сохраняем мы объект `fig`). В первом мы строим ВАХ диода, подписываем оси и выставляем пределы по осям. Во втором мы делаем то же самое, но для логарифмированных значений силы тока. Отдельно выделим, что некоторые методы объекта `ax` имеют иные имена

[9] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.subplots.html

относительно `plt`:

`plt.title` → `ax.set_title`,
`plt.xlim` → `ax.set_xlim`,
`plt.xlabel` → `ax.set_xlabel`.

Полный список методов представлен в документации[10]. Простое правило определения, изменилось ли имя таково: если конкретный метод рисует точки или линии, то имя не изменилось. Например, `plt.scatter` → `ax.scatter`. Если же метод устанавливает некоторый параметр рисунка, то его имя меняется.

Внимательный читатель заметит странную конструкцию в строке 12: `'ln I/I$_0$'`. Так можно добавить верхний или нижний индекс, состоящий из одного символа, аналогично описанному в разделе 3.5.

Вызов `ax.set_(xy)lim` можно сократить, не указывая имён аргументов, если указывать и верхний, и нижний пределы. Запись `(xy)` означает, что есть две разные функции для оси абсцисс и оси ординат, имена которых отличаются одной буквой в названии.

Можно перейти к неявному подходу, определив перед этим рисунок `fig` и оси `axes` (явный подход). Тогда `plt` будет использовать последний элемент из списка `axes`.

Можно перейти к явному подходу, неявно определив рисунок, вызовом `ax = plt.gca()` (gca = **g**et **c**urrent **a**xis) и дальше работать уже с объектом `ax`.

Изменим рисунок, объединив оси абсцисс. Для этого явно укажем в первой строке, что эти оси должны быть общие. При этом уберем вызов `ax1.set_xlim`: пределы достаточно задать для одних осей. Также уберем вызов `ax1.set_xlabel`: подпись должна остаться только у нижних осей.

```
1 fig, axes = plt.subplots(2, 1, figsize=(5, 8), sharex=True,
  ↪  gridspec_kw=dict(hspace=0.05))
```

Необязательный аргумент `gridspec_kw` в вызове `plt.subplots` определяет параметры расположения осей на рисунке. В данном примере мы только изменяем расстояние между верхними и нижними осями,

[10] https://matplotlib.org/stable/api/axes_api.html

выставляя его равным 5% от высоты осей. Если бы оси располагались горизонтально, то мы изменяли бы параметр `wspace`. В случае двумерной сетки из осей имеет смысл определять оба. Более подробно можно прочитать в документации[11].

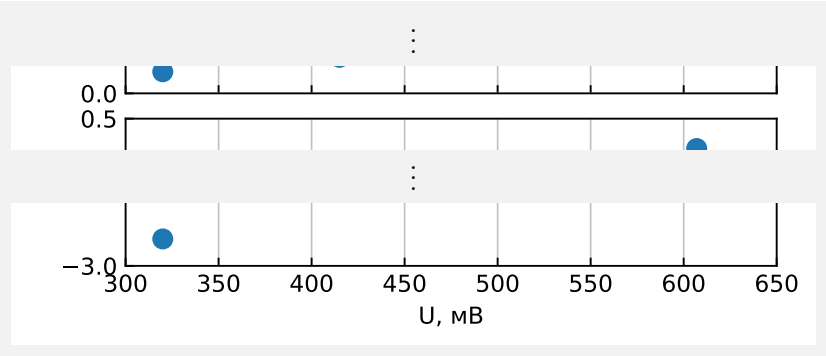


Рис. 3.10: Объединение осей на рисунке 3.9

3.7 Линейная регрессия и построение графика

Регрессионный анализ — это набор статистических методов исследования влияния одной или нескольких независимых переменных $x_1, x_2, \ldots$ на некоторую зависимую переменную y. Терминология зависимых и независимых переменных отражает лишь математическую зависимость переменных, а не причинно-следственные отношения. Простейшим примером независимой и зависимой переменных, соответственно, будут время t и координата x при анализе зависимости $x(t)$ координаты от времени в разделе 3.3. Наиболее распространённый вид регрессионного анализа — линейная регрессия, когда находят линейную функцию, которая, согласно определённым математическим критериям, наиболее соответствует данным.

При этом в список целей регрессионного анализа включают предсказание значения зависимой переменной с помощью независимой(-ых) и определение вклада отдельных независимых переменных в вариацию зависимой. Таким образом, при проведении такого анализа необходимо предположение о том, какая зависимость связывает разные переменные.

Далее будет использован метод наименьших квадратов (МНК). Метод предназначен для аппроксимации набора экспериментальных точек,

[11] https://matplotlib.org/stable/api/_as_gen/matplotlib.gridspec.GridSpec.html

содержащих погрешности, “хорошей” функцией. Предложен К. Ф. Гауссом и А. М. Лежандром и первоначально применялся для обработки результатов астрономических и геодезических наблюдений. Строгое обоснование метода и нахождение границ его применимости даны А. А. Марковым и А. Н. Колмогоровым.

Обозначим набор из N экспериментальных точек $\{x_i, y_i\}$, а аппроксимирующую их функцию (модельную функцию) — $f(x)$. В силу того, что эта функция должна описывать результаты эксперимента не точно, а приближенно, $f(x_i) \neq y_i$. Идея Гаусса заключалась в том, что “убыток” от замены точного значения y_i на приближенное $f(x_i)$ пропорционален квадрату их расхождения $(f(x_i) - y_i)^2$; тогда в качестве оптимальной функции $f(x)$ логично принять такую, которая дает минимальный “суммарный убыток”:

$$\chi^2 = \sum_{i=1}^{N} (f(x_i) - y_i)^2 = \min .$$

Этого можно добиться подбором параметров функции $f(x)$. Подчеркнем, что вид самой функции $f(x)$ нужно выбрать заранее!

3.7.1 Случай пропорциональной зависимости

Вернемся к задаче 102 (раздел 3.5). Глядя на получившийся график, прямо хочется провести через экспериментальные точки прямую линию. В то же время мы знаем, что эта прямая должна пройти через точку (0,0). Таким образом, перед нами классический случай прямой пропорциональности $y = f(x) = ax$.

В таком случае для того, чтобы найти константу a и её дисперсию S_a^2, произведем следующие вычисления [9]. Далее всюду будем обозначать независимую переменную x, а зависимую — y. Индексом i обозначим номер измерения. Рассмотрим три характерных случая.

1. Если дисперсии $\{\sigma_i^2\}$ для величин $\{y_i\}$ известны, то

$$a = \frac{\sum\limits_{i=1}^{N} \frac{x_i y_i}{\sigma_i^2}}{\sum\limits_{i=1}^{N} \frac{x_i^2}{\sigma_i^2}}, \qquad S_a^2 = \frac{1}{\sum\limits_{i=1}^{N} \frac{x_i^2}{\sigma_i^2}}. \tag{3.1}$$

2. Если дисперсии $\{\sigma_i^2\}$ для величин $\{y_i\}$ известны и одинаковы ($\{\sigma_i^2\} = \{\sigma_0^2\}$), то

$$a = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2}, \qquad S_a^2 = \frac{\sigma_0^2}{\sum_{i=1}^{N} x_i^2}. \tag{3.2}$$

3. Если дисперсии $\{\sigma_i^2\}$ для величин $\{y_i\}$ одинаковы, но неизвестны, то можно оценить дисперсию S_a^2 следующим образом:

$$S_a^2 = \frac{\sum_{i=1}^{N} x_i^2 \cdot \sum_{i=1}^{N} y_i^2 - \left(\sum_{i=1}^{N} x_i y_i\right)^2}{(n-1) \cdot \left(\sum_{i=1}^{N} x_i^2\right)^2} \tag{3.3}$$

Например, имеется зависимость $v^2(x)$ квадрата скорости тележки для трёх разных наклонов стола, содержащаяся в списках `x`, `v2_1`, `v2_2` и `v2_3`. Допустим, вычисленное стандартное отклонение квадрата скорости $\sigma_{v^2} = 1.5 \cdot 10^{-7} \frac{\text{м}^2}{\text{с}^2}$ и одинаково для всех трех случаев. Это основание воспользоваться алгоритмом, описанным формулами (3.2).

Переименуем `v2_1` в `y`, чтобы не запутаться в обозначениях.

```
# начнем с первого наклона стола,
# потом повторим для v2_2 и v2_3
y = v2_1       # м / с
s0 = 1.5e-7    # м^2 / с^2

a = np.sum(x * y) / np.sum(x**2)
Sa = np.sqrt( s0**2 / (np.sum(x**2))**2 )
print(a, Sa)
>>> 0.04081818181818182 6.818181818181817e-08
```

Проверить соответствие результатов эксперимента и предлагаемой линейной модели можно, воспользовавшись коэффициентом детерминации R^2, являющимся квадратом коэффициента взаимной корреляции r. Мы можем его определить из отношения суммы квадратов остатков регрессии SS_{res} и общей суммы квадратов SS_{tot}. Если обозначить предсказываемые (рассчитанные в рамках модели линейной зависимости)

значения величины y как $\hat{y}$, а среднее арифметическое от экспериментальных значений — как $\bar{y}$, то

$$SS_{res} = \sum_{i=1}^{N}(y_i - \hat{y}_i)^2, \quad \hat{y}_i = ax_i,$$
$$SS_{tot} = \sum_{i=1}^{N}(y_i - \bar{y})^2, \quad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i,$$
$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}. \tag{3.4}$$

Проведем вычисления:

```
ym = np.mean(y)      # среднее арифметическое (англ. mean)
SSres = np.sum((y - a*x)**2)
SStot = np.sum((y - ym)**2)
R2 = 1 - SSres / SStot
print(R2)
>>> 0.9663928082395666
```

Теперь с точки зрения разумности количества труда объединим все расчеты в один цикл и выведем результаты на график для того, чтобы их было проще воспринимать. Вставим в пример из раздела 3.5 на стр. 25 между строками 3 и 4 следующее:

```
xx = np.linspace(0, 1.1, 100)
for y in [v2_1, v2_2, v2_3]:
    # расчет a, Sa, R2 - см. выше
    plt.plot(xx, a*xx, label=f'{a= :.2e} $\pm$ {Sa:.2e};
    ↪  $R^2$= {R2:.3f}')
```

Разберем это дополнение построчно.

1. Подготовка массива координат для последующего рисования прямых.
2. Цикл, проходящий по всем различным наборам квадратов скоростей $\{v_i^2\}$.
3. Расчет коэффициента наклона прямой, его стандартного отклонения и коэффициента детерминации — разобран выше.
4. Рисование прямой линии.

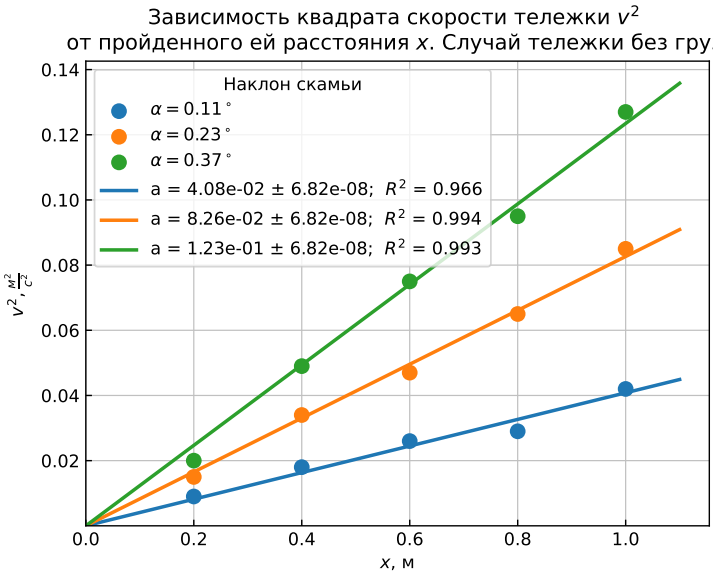


Рис. 3.11: Результат построения нескольких графиков и линейных аппроксимаций соответствующих экспериментальных данных

В результате получим следующий рисунок. Обсудим отдельные детали более подробно.

В строке 1 `np.linspace(start, stop, num)` создает массив, состоящий из `num` значений, равномерно распределенных между `start` и `stop`. Этот массив включает в себя значения как `start`, так и `stop`. Не стоит брать значение `num` больше 1000, если только рисуемая кривая не осциллирует быстро. Здесь мы возьмем `num = 100`.

В строке 2 конструкция из двух ключевых слов `for ... in ...:` обозначает цикл. В нем переменной слева от `in` поочередно присваивается каждый элемент коллекции справа от `in`, пока та не кончится. При этом тело цикла (все повторяемые инструкции) должно иметь отступ в начале строки относительно этой строки (см. на строки 3–5). Более подробно разобрано, например, в [3], раздел 8.

В строке 4 `plt.plot(x, y)` рисует ломаную линию, соединяющую по порядку точки (x_i, y_i). Её стиль можно задать аргументом `fmt` аналогично написанному в разделе 3.4. В данном случае с помощью этого метода мы рисуем прямые с рассчитанными заранее параметрами. Подробнее можно прочитать в документации[12]. Значение аргумента

[12] https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

`label` в данном конкретном случае — это форматированная строка (f-string). Она отличается от обычных строк буквой f перед первой кавычкой: `f'...'` и дает более удобный интерфейс для подставления переменных. Например, здесь мы выводим значение переменной `R2` с помощью конструкции `{R2 = }`. Более подробно о них можно узнать, например, в [3], раздел 7 или в документации к языку[13].

Очевидно, в таком виде этот график является техническим, для представления не годным: легенда накладывается на линии. Тем не менее, с него удобно перенести данные, необходимые для дальнейшей работы, а после убрать чересчур длинные подписи.

Обратим внимание, что цвета маркеров `plt.scatter` и ломаных линий `plt.plot` меняются по отдельности (мы всё так же не задаем их явно), что в данном конкретном случае очень удобно.

При необходимости можно изменить набор делений на любой оси при помощи методов `plt.xticks` и `plt.yticks` (или `ax.set_(xy)ticks`), передав им новый список делений где-либо перед сохранением графика в файл. Например,

```
1 plt.xticks([0, 0.25, 0.5, 0.75, 1])
```

изменит ось абсцисс, добавив на ось только те деления, что соответствуют значениям в передаваемом массиве и убрав все остальные. Это будет выглядеть так:

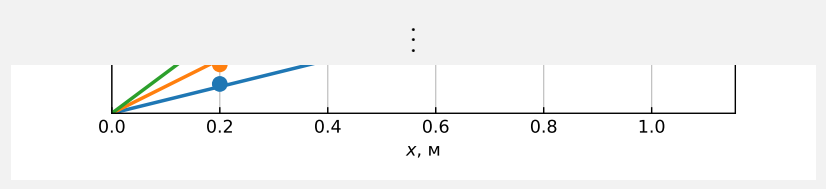


Рис. 3.12: Исправленная ось абсцисс рисунка 3.11

Альтернативно можно задать список значений при помощи функции[14] Numpy `np.arange(start, stop, step)`, которая вернет массив значений `[start, start+step, ···, stop)` с шагом `step`, не включающий правую границу `stop`. Эта функция является аналогом встроенной в Python функции `range`, но позволяет использовать вещественные числа.

[13] https://docs.python.org/3/tutorial/inputoutput.html
[14] https://numpy.org/doc/stable/reference/generated/numpy.arange.html

```
np.arange(0, 1, 0.2)    # array([0. , 0.2, 0.4, 0.6, 0.8])
```

Возможна и более тонкая настройка делений на оси при помощи локаторов и форматтеров[a].

[a] https://matplotlib.org/stable/api/ticker_api.html

3.7.2 Случай линейной зависимости

Рассмотрим заново пример задачи 103 из раздела 3.4. Зависимость $f(m)$ должна быть линейной, однако на этот раз она отнюдь не обязана проходить через точку (0, 0). Это более общий случай линейной зависимости с ненулевой константой $y = f(x) = ax + b$.

С одной стороны, можно подойти к вопросу нахождения постоянных a и b аналогично предыдущему разделу. Пособие [9] содержит в себе всё необходимое для их вычисления. С другой стороны, хочется получить на руки некий универсальный метод регрессии экспериментальных данных.

Воспользуемся библиотекой Scipy. Общий подход будет следующий: определим модельную функцию, а после минимизируем отклонение вычисленных значений этой функции от экспериментальных данных путем варьирования её параметров. Возьмем пример из раздела 3.4 на стр. 22 и вставим между 5 и 6 строками (между определением данных и рисованием графика) следующее:

```
from scipy.optimize import curve_fit

x, y, dy = m, f, df

def func(x, a, b):
    return a * x + b

popt, pcov = curve_fit(func, x, y,
    p0=[y[0]/x[0], y[0]], sigma=dy, absolute_sigma=True)
a, b = popt
Sa, Sb = np.sqrt(np.diag(pcov))
print(f'{a = :f} +- {Sa:f}')
print(f'{b = :f} +- {Sb:f}')
>>> a = 0.232018 +- 0.105140
>>> b = 0.028635 +- 0.001943
```

```
13 xx = np.linspace(0, 0.03, 100)
14 plt.plot(xx, a*xx+b, zorder=1000)
```

Разберем это дополнение построчно.

1. Импорт функции `curve_fit` из библиотеки Scipy. Она предназначена для того, чтобы подогнать некоторую заданную нами модельную функцию к заданным точкам (в нашем случае, экспериментальным) методом наименьших квадратов.
2. Переименование массивов `{m, f, df}` → `{x, y, dy}` для того, чтобы не запутаться в обозначениях.

3–4. Объявление модельной функции.

5–6. Расчет параметров модельной функции с помощью `curve_fit`.

7. Помещение вычисленных параметров модельной функции в соответствующие переменные.
8. Вычисление их стандартных отклонений.

9–12. Вывод получившихся значений.

13–14. Построение графика.

В строке 3 определяется модельная функция $f(x, a, b) = ax + b$ одной независимой переменной x и двух параметров a и b. Важно, чтобы независимая переменная x была в списке аргументов первой.

Вся магия происходит в строках 5 и 6, во время вызова `curve_fit`. Разберем передаваемые аргументы. Первые три — это обязательные параметры: модельная функция, а также значения независимой и зависимой переменных, соответственно. Остальные могут быть, в принципе, опущены. Перечислим их.

- `p0` — список начальных значений каждого из параметров, в том же порядке, в каком они определены в модельной функции. Они необходимы алгоритму в качестве стартовой точки для вычислений: не всегда значения по умолчанию хороши, и в ряде случаев верный ответ не может быть с ними достигнут, особенно в случае нелинейной регрессии. В данном случае параметров два: наклон прямой и точка пересечения с осью ординат, a и b. Неплохим стартовым значением для наклона будет y_1/x_1, а для пересечения — значение y_1.
- `sigma` — список стандартных отклонений для y.
- `absolute_sigma` — флаг, определяющий, использовать ли значения из предыдущего аргумента по их абсолютной величине. Если не указать или выставить `False`, то значения стандартных отклонений не будут учтены.

`curve_fit` возвращает два объекта. Первый, `popt`, содержит наилучшие значения параметров в том же порядке, как они входят в определение модельной функции. Второй, `pcov`, содержит в себе матрицу ковариаций параметров. Мы можем использовать её для определения стандартных отклонений параметров: известно, что на её диагонали стоят дисперсии параметров, корень квадратный из которых и дает стандартные отклонения. Именно это мы и делаем в строке 8.

После этого в строках 9–14 мы выводим на экран подогнанные в хорошем смысле этого слова значения параметров и строим прямую согласно им.

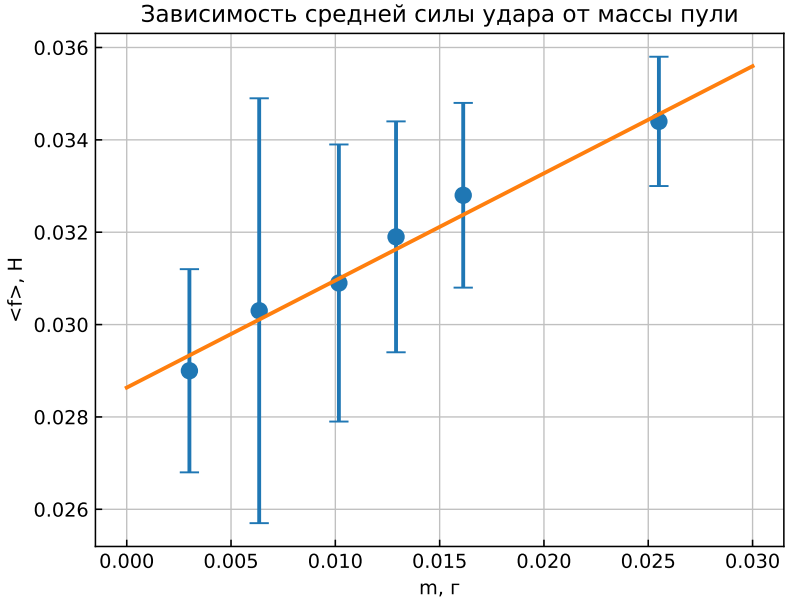


Рис. 3.13: Результат построения графика и линейной аппроксимации соответствующих экспериментальных данных

Оценить степень соответствия экспериментальных данных вычисленной модели можно при помощи коэффициента взаимной корреляции r. В отличие от предыдущей модели, здесь коэффициент детерминации R^2 в таком виде, как мы его определили раньше, не работает. Можно показать, что r находится в диапазоне от -1 до 1, причем $r = 1$ тогда и только тогда, если все экспериментальные точки лежат на одной прямой. Значение $|r| \approx 0.9 \div 0.95$ говорит о неплохом согласии между моделью и экспериментальными данными. Воспользуемся выражением из пособия [9].

$$r = \frac{\sum_{i=1}^{N} ((x_i - \bar{x}) \cdot (y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{N} (x_i - \bar{x})^2 \cdot \sum_{i=1}^{N} (y_i - \bar{y})^2}}, \quad \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i, \quad \bar{y} = \frac{1}{N} \sum_{i=1}^{N} y_i. \tag{3.5}$$

Вычислим его для нашего случая:

```
xm, ym = np.mean(x), np.mean(y)
r = np.sum((x-xm)*(y-ym)) / np.sqrt(np.sum((x-xm)**2) *
 ↪  np.sum((y-ym)**2))
print(f'{r = :f}')
>>> r = 0.988887
```

Коэффициент взаимной корреляции, близкий к 0.99, указывает нам на то, что выбор линейной модели является оправданным.

3.8 Нелинейная регрессия и построение графика

Кратко рассмотрим, как провести анализ системы с нелинейной зависимостью. В качестве примера возьмем задачу 118, в которой изучается пружинный маятник, чей подвес может колебаться в вертикальной плоскости с разной частотой. Пусть у нас есть измеренная амплитудно-частотная характеристика (АЧХ) такого маятника $A(\omega)$. Известен теоретический вид этой зависимости,

$$A(\omega) = \frac{f_0}{\sqrt{(\omega_0 - \omega)^2 + 4\delta^2\omega^2}}, \tag{3.6}$$

где параметрами являются $f_0 = F_0/m$ — приведенная амплитуда силы, ω_0 — собственная частота и δ — коэффициент затухания.

Приведем расчет параметров АЧХ, опуская все принципиально неважные детали. Предположим, что значения частот и соответствующих им амплитуд уже находятся в списках `x` и `y`, а дисперсии амплитуд неизвестны и одинаковы.

```
def amp(w, w0, delta, f0):   # модельная функция A(ω; ω0, δ, f0)
    return f0 / np.sqrt( (w0**2 - w**2)**2 + 4 * delta**2 *
     ↪  w**2 )
```

```
popt, pcov = curve_fit(amp, x, y, p0=(3.5, 0.5, 1))
w0, delta, f0 = popt
Sw0, Sdelta, Sf0 = np.sqrt(np.diag(pcov))
print(f'{   w0 = : f} +- {Sw0:f}')
print(f'{delta = : f} +- {Sdelta:f}')
print(f'{   f0 = : f} +- {Sf0:f}')
>>>    w0 =  3.490206 +- 0.002606
>>> delta =  0.072659 +- 0.003098
>>>    f0 =  0.445524 +- 0.012464

xx = np.linspace(3, 4, 200)
plt.scatter(x, y, color='k')
plt.plot(xx, amp(xx, *popt), color='r')
```

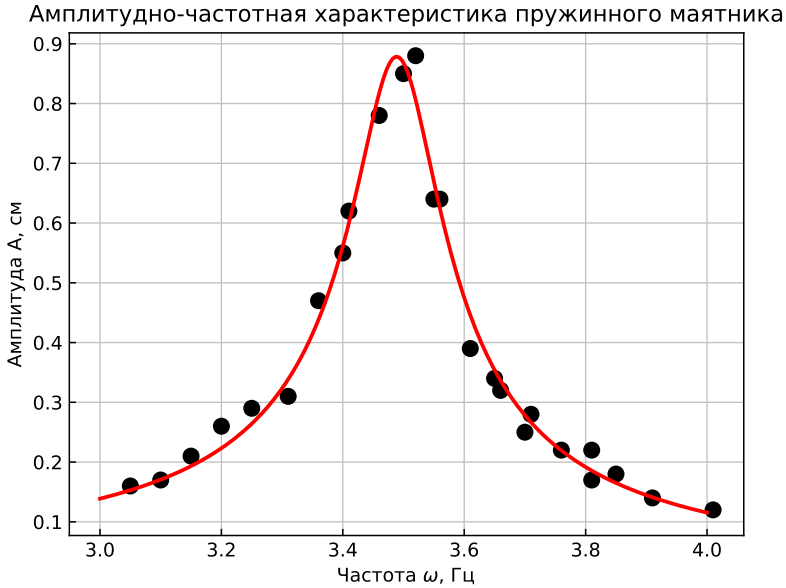


Рис. 3.14: Результат построения графика и нелинейной аппроксимации соответствующих экспериментальных данных

К сожалению, в случае нелинейной регрессии коэффициент взаимной корреляции теряет смысл.

3.9 График в полярной системе координат

Рассмотрим в качестве примера задачу 116 про определение тензора инерции твердого тела. При ее выполнении необходимо экспериментально определить сечение эллипсоида инерции некоторого тела и сравнить его с рассчитанным. Это можно сделать при помощи графика в полярных координатах.

Опустим все несущественные детали, подробно разобранные выше. Предположим, что список `alpha` содержит углы от 0 до 180 градусов в радианах, а `J` и `Jt` — соответствующие им значения момента инерции, экспериментальные и расчетные, соответственно. В самом начале перейдем от декартовой систем координат к полярной, поменяв проекцию. Теперь вместо осей (X, Y) мы используем оси (Θ, R).

```
1  fig, ax = plt.subplots(subplot_kw={'projection': 'polar'})

2  ax.plot(alpha, J, 'o', color='C0', label='эксперимент')
3  ax.plot(alpha + 3.14, J, 'o', color='C0')
4  ax.plot(alpha, Jt, 'o', color='C1', label='расчёт')
5  ax.plot(alpha + 3.14, Jt, 'o', color='C1')

6  ax.set_rlim(0, 80)            # Выставим пределы по оси R
7  ax.set_rticks([20, 40, 60])   # Меньше делений
8  ax.set_rlabel_position(-90)   # Угол для подписей к делениям
9  ax.grid(True)                 # Включаем сетку
10 ax.legend(loc=(0.8,0.94))     # Отображаем легенду
```

Обратите внимание на строку 10, в которой создается легенда. Передаваемый аргумент `loc` содержит координаты нижнего левого угла рамки легенды. При этом точке (0,0) соответствует нижний левый угол рисунка, а точке (1,1) — верхний правый.

Иначе в качестве значения `loc` можно выбрать одно из следующего списка:

`'upper left'`, `'upper center'`, `'upper right'`,
`'center left'`, `'center'`, `'center right'`,
`'lower left'`, `'lower center'`, `'lower right'`

или `'best'` (значение по умолчанию).

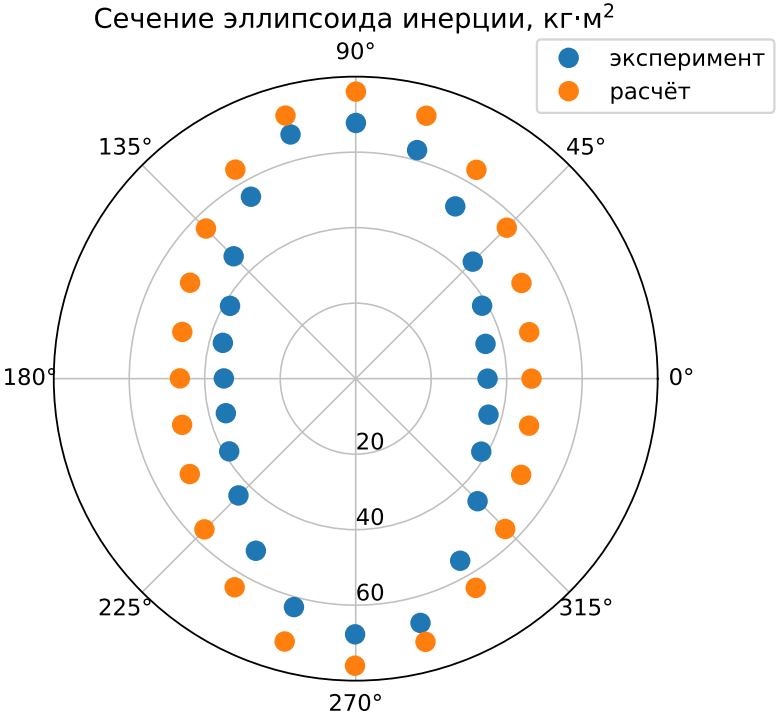


Рис. 3.15: Результат построения графика в полярной системе координат

Глава 4. Дальнейшее изучение

В качестве литературы для дальнейшего изучения мы можем порекомендовать следующее.

Во-первых, как у языка, так и у каждой из библиотек есть как хорошая документация, так и руководство пользователя (на английском языке).

- Python: https://docs.python.org/3/
- Numpy: https://numpy.org/doc/stable/
- Matplotlib: https://matplotlib.org/stable/index.html
- Scipy: https://docs.scipy.org/doc/scipy/

Во-вторых, на русском языке рекомендуем книги [1—6]. По сложности их можно отнести к базовым. Их преимущество в том, что они последовательно и подробно выдают материал, переходя от простого к сложному.

Далее отдельно можно отметить несколько книг, охватывающих существенно бóльший пласт материала. При этом подача материала в них рассчитана на читателя, обладающего некоторым опытом.

1. “Python и анализ данных” [10] рассказывает об обработке и анализе данных при помощи библиотеки Pandas.
2. “Python Polars: подробное руководство” [11] рассказывает об обработке и анализе данных при помощи библиотеки Polars, наследнике Pandas.
3. “Элегантный Scipy” [12] описывает углубленное применение Scipy для решения различных задач.
4. “Использование библиотек Python в научных исследованиях” [13] описывает работу с библиотеками Numpy, Matplotlib и Sympy.
5. “Numerical Python” [14] является большим сборником всевозможных эффективных подходов к решению большого спектра разных задач (на англ. языке).
6. “Сверхбыстрый Python” [15] рассказывает о возможных путях ускорения выполнения программ с акцентом на большие данные.

Наконец, необходимо признать, что на Matplotlib свет клином не сошелся. Эта библиотека является наиболее распространенной в силу устоявшегося интерфейса и стабильной работы. Кроме того, основные принципы работы с ней аналогичны другим библиотекам. Однако она не лишена недостатков. Основными из них можно назвать слабую интерактивность, что может быть важно для работы “здесь и сейчас”, а так же не очень проработанный интерфейс работы с трехмерными графиками. Поэтому мы приведем здесь список других библиотек

вместе с краткими описаниями, которые могут быть полезны в том или ином случае.

1. Plotly[1] — отличная интерактивная библиотека, заточенная на представление результатов в браузере. Хорошо реализована трехмерная графика.
2. Seaborn[2] — надстройка над Matplotlib, заточенная на работу со статистическими данными.
3. Bokeh[3] — интерактивная библиотека, хорошо работающая с достаточно большими наборами данных.
4. Altair[4] — реализует иной, декларативный, подход к описанию графики.

Важно помнить, что для каждого вида деятельности есть подходящие и неподходящие инструменты. Поэтому выбирать их необходимо, основываясь на удобстве работы и предоставляемых возможностях, а также ваших знаниях и умении с этими инструментами работать.

[1] https://plotly.com/python/
[2] https://seaborn.pydata.org/
[3] https://docs.bokeh.org/en/latest/
[4] https://altair-viz.github.io/index.html

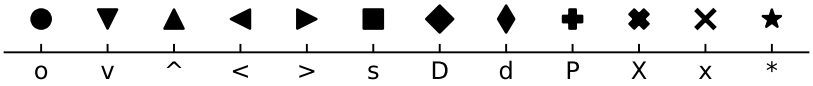
Приложения

А Краткая справка

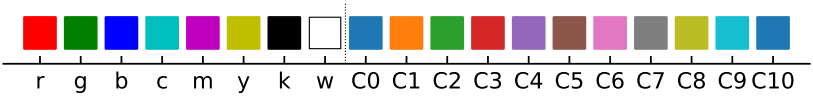
```
import numpy as np
import matplotlib.pyplot as plt # импорт библиотек
plt.style.use('ff.mpl.style')   # подключение стиля

# Подготовка данных
# Создание рисунка, осей и т.п.
# Отрисовка данных
# Настройка рисунка
# Вывод на экран/в файл
```

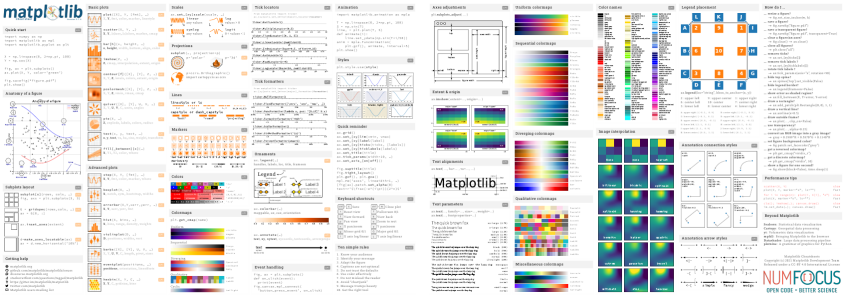
Маркеры



Цвета



Шпаргалка от разработчиков Matplotlib[1]



[1] https://matplotlib.org/cheatsheets/cheatsheets.pdf

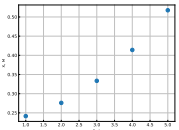
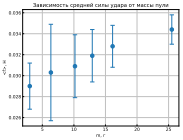
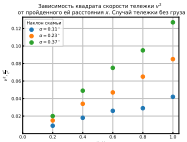
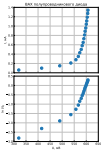
	Неявный подход `plt`[2]	Явный подход `fig`[3] и `ax`[4]	
Точечный график	`plt.scatter(x, y, ...)`	`ax.scatter(...)`	[5]
Точечный график с погрешностями	`plt.errorbar(x, y, yerr, ...)`	`ax.scatter(...)`	[6]
Ломаная линия	`plt.plot(x, y, fmt, ...)`	`ax.plot(...)`	[7]
Подпись к графику	`plt.title('Название')`	`ax.set_title(...)`	[8]
Подписи к осям	`plt.(xy)label('Величина X, размерность')`[9]	`ax.set_(xy)label(...)`	[10]
Координатная сетка	`plt.grid(True/False)`	`ax.grid(...)`	[11]
Пределы по осям	`plt.(xy)lim(xmin, xmax)`	`ax.set_(xy)lim(...)`	[12]
Отметки на осях	`plt.(xy)ticks([0,1,2,3])`	`ax.set_(xy)ticks(...)`	[13]
Легенда	`plt.legend()`	`ax.legend(...)`	[14]
Запись в файл	`plt.savefig('Имя файла', bbox_inches='tight')`	`fig.savefig(...)`	[15]

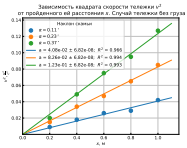
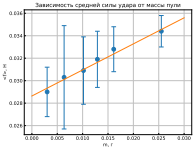
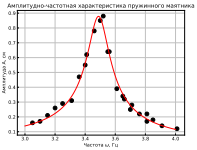
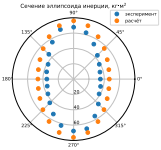
[2] https://matplotlib.org/stable/api/pyplot_summary.html
[3] https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.html
[4] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.html
[5] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.scatter.html
[6] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.errorbar.html
[7] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.plot.html
[8] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_title.html
[9] Запись `(xy)` означает, что есть две функции для оси абсцисс и оси ординат, имена которых отличаются одной буквой в названии.
[10] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xlabel.html
[11] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.grid.html
[12] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xlim.html
[13] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.set_xticks.html
[14] https://matplotlib.org/stable/api/_as_gen/matplotlib.axes.Axes.legend.html
[15] https://matplotlib.org/stable/api/_as_gen/matplotlib.figure.Figure.savefig.html

Б Листинги примеров

Все файлы можно найти по ссылке ниже.

http://genphys.phys.msu.ru/rus/sci/nanogroup/lectures_python-plots.html

Имя файла	Раздел	Рисунок
`environment.yml`	1.3.1	—
`ff.mplstyle`	2	—
`example_1_simple.py`	3.3	
`example_2_errors.py`	3.4	
`example_3_sameaxis.py`	3.5	
`example_4_multipleaxes.py`	3.6	

Имя файла	Раздел	Рисунок
`example_5_proportionality.py`	3.7.1	
`example_6_linear.py`	3.7.2	
`example_7_nonlinear.py`	3.8	
`example_8_polar.py`	3.9	

В Описание стилевого файла `ff.mplstyle`

```
1  lines.linewidth:            2
2  lines.markersize:           8
3  errorbar.capsize:           5
4  axes.grid:                  True
5  polaraxes.grid:             True
6  grid.color:                 "#c0c0c0"
7  grid.linewidth:             0.7
8  axes.axisbelow:             True
9  axes.formatter.useoffset:   False
10 xtick.direction:            in
11 ytick.direction:            in
```

Разберем построчно:

1. Толщина линии (пункты, пт), создаваемой `plt.plot`.
2. Размер маркеров (пт), создаваемых `plt.scatter`.
3. Длина линий (пиксели) на концах отрезков погрешностей, создаваемых `plt.errorbar`.

4–5. Наличие сетки в декартовых и полярных координатах.

6–7. Цвет линий сетки, их толщина (пт).

8. Настройка положения линий осей в виртуальной стопке.
9. Использование смещения для подписей, когда диапазон данных мал по сравнению с минимальным абсолютным значением данных.

10–11. Направление отметок на осях абсцисс и ординат: внутрь рамки осей.

Более подробное описание этих и других опций можно найти в документации[16].

[16] https://matplotlib.org/stable/users/explain/customizing.html

Литература

1. Седер, Н. *Python. Экспресс-курс* 3 издание. 480 с. (Питер, 2019).
2. Задорожный, С. С., Фадеев, Е. П. *Объектно-ориентированное программирование на языке Python.* 40 с. https://cmp.phys.msu.su/sites/default/files/%D0%9E%D0%9E%D0%9F_%D0%BD%D0%B0_Python_%D0%A3%D1%87%D0%B5%D0%B1%D0%BD%D0%BE%D0%B5%20%D0%BF%D0%BE%D1%81%D0%BE%D0%B1%D0%B8%D0%B5_var7.pdf (Физический фак-т МГУ им. М. В. Ломоносова, Москва, 2022).
3. Шипило, Д. Е., Коновко, А. А., Лукашёв, А. А., Панов, Н. А. *Язык программирования Python. Семестр 3 — после C++.* 99 с. https://cmp.phys.msu.ru/sites/default/files/Python_after_Cpp.pdf (Физический фак-т МГУ им. М. В. Ломоносова, Москва, 2024).
4. Лутц, М. *Изучаем Python* 5 издание. 2 т. (Диалектика, 2019).
5. Лутц, М. *Программирование на Python* 4 издание. 2 т. (Диалектика-Вильямс, 2024).
6. Саммерфилд, М. *Программирование на Python 3. Подробное руководство* 4 издание. 608 с. (Символ-Плюс, 2020).
7. Лутц, М. *Python. Карманный справочник* 5 издание. 320 с. (Вильямс, 2015).
8. Ананьева, Н. Г., Ананьева, М. С., Самойлов, В. Н. *Графическое оформление результатов эксперимента. Построение графика в прямоугольной системе координат.* 23 с. http://genphys.phys.msu.ru/rus/lab/vtek/Graf_of_rez_eksp_2016.pdf (Физический фак-т МГУ им. М. В. Ломоносова, Москва, 2016).
9. Митин, И. В., Русаков, В. С. *Анализ и обработка экспериментальных данных. Учебно-методическое пособие для студентов младших курсов.* 4 издание. 44 с. (Физический фак-т МГУ им. М. В. Ломоносова, Москва, 2009).
10. Маккинни, У. *Python и анализ данных* 3 издание. 536 с. (ДМК Пресс, 2023).
11. Янссенс, Й., Ньюдорп, Т. *Python Polars: подробное руководство.* 502 с. (Books.kz, 2025).
12. Нуньес-Иглесиас, Х., Уолт, Ш., Дэшноу, Х. *Элегантный Scipy.* 266 с. (ДМК Пресс, 2018).
13. Пылькин, А. Н., Соколова, Ю. С. *Использование библиотек Python в научных исследованиях.* 236 с. (Горячая Линия - Телеком, 2025).

14. Johansson, R. *Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib* (Apress, 2019).

15. Антао, Т. *Сверхбыстрый Python. Эффективные техники для работы с большими наборами данных.* 370 с. (ДМК Пресс, 2023).

Предметный указатель

f-строки, 37

Gridspec, 32

MathText, 26
matplotlibrc, 14, 52

zorder, 18

Виды графиков
    ломаная линия, 36
    точечный, 21
Изменение делений на оси, 38
Коэффициент детерминации, 35, 41
Легенда, 28, 44
Название графика, 24
Общие оси, 32
Погрешность, 23
Подписи к осям, 23
Подход
    неявный, 21
    смена, 32
    явный, 29
Поля графика, 29
Пределы значений по осям, 27
Работа с отдельными осями, 31
Размер рисунка, 30
Состав рисунка, 18
Сохранение в файл, 22
Строка формата, 24
Структурирование кода, 23
Цвета, 28