

Министерство науки и высшего образования Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО

Институт физики и математики
Кафедра физики
Кафедра высшей математики

М. В. Сысоева И. В. Сысоев

АНАЛИЗ И СТАТИСТИЧЕСКАЯ ОБРАБОТКА ДАННЫХ ФИЗИЧЕСКИХ ЭКСПЕРИМЕНТОВ В УСЛОВИЯХ ПОМЕХ

Учебное пособие



ПОЛИТЕХ-ПРЕСС

Санкт-Петербургский
политехнический университет
Петра Великого

Санкт-Петербург
2026

УДК 53.023(075.8)

ББК 22.1я73

С95

Р е ц е н з е н т ы:

Доктор физико-математических наук, доцент, заведующий кафедрой систем автоматизированного проектирования Санкт-Петербургского государственного электротехнического университета «ЛЭТИ» *Д. Н. Бутусов*

Доктор физико-математических наук, доцент, директор Института физики и математики Санкт-Петербургского политехнического университета Петра Великого *П. В. Захаров*

Сысоева М. В. Анализ и статистическая обработка данных физических экспериментов в условиях помех : учеб. пособие / М. В. Сысоева, И. В. Сысоев. — СПб. : ПОЛИТЕХ-ПРЕСС, 2026. — 75 с.

В учебном пособии приводятся необходимые сведения из теории обработки сигналов и данных, в том числе из статистической радиофизики, теории приближений и математической статистики, описывается методика, даются алгоритмы и код программ для обработки на примере конкретных лабораторных работ практикума по общей физике.

Предназначено для студентов 1–3 курсов физико-математических и инженерных специальностей и направлений подготовки (бакалавров и специалистов) широкого профиля, в том числе экспериментальных программ и программ с углубленным изучением (инженерный авангард) фундаментальных дисциплин.

Табл. 7. Ил. 9. Библиогр.: 4 назв.

Печатается по решению

Совета по издательской деятельности Ученого совета

Санкт-Петербургского политехнического университета Петра Великого.

ISBN 978-5-7422-9506-8

doi:10.18720/SPBPU/2/id26-441

© Сысоева М. В., Сысоев И. В., 2026

© Санкт-Петербургский политехнический университет Петра Великого, 2026

Ministry of science and higher education of the Russian Federation

PETER THE GREAT

ST. PETERSBURG POLYTECHNIC UNIVERSITY

Institute of Physics and Mathematics

Department of Physics

Department of Higher Mathematics

M. V. Sysoeva I. V. Sysoev

ANALYSIS
AND STATISTICAL
PROCESSING OF DATA
FROM EXPERIMENTS
IN PHYSICS IN PRESENCE
OF INTERFERENCE

Training manual



POLYTECH PRESS

Peter the Great
St. Petersburg Polytechnic
University

Saint Petersburg

2026

Reviews:

Doctor in Physics and Mathematics, Head of Department of Computer Aided Design of St. Petersburg Electrotechnical University “LETI”

D. N. Butusov

Doctor in Physics and Mathematics, Director of Institute of Physics and Mathematics of Peter the Great St. Petersburg Polytechnic University

P. V. Zakharov

Sysoeva M. V. **Analysis and statistical processing of data from experiments in physics in presence of interference** : Training manual / M. V. Sysoeva, I. V. Sysoev. — St. Petersburg : POLYTECH-PRESS, 2026. — 75 p.

The tutorial provides the useful information from the signal and data processing theory, including statistical radiophysics, approximation theory and mathematical statistics, describes the methodology, provides algorithms and program code based on specific laboratory works in general physics.

It is intended for students of 1–3 years of education of physics, mathematics and engineering programs (bachelors and engineers) of a wide range, including experimental and advanced study programs.

Tables. 7. Figures. 9. References.: 4 titles.

Printed by the Publishing Council of the Peter the Great St. Petersburg polytechnic university Academic Council.

ISBN 978-5-7422-9506-8

doi:10.18720/SPBPU/2/id26-441

© Sysoeva M. V., Sysoev I. V., 2026

© Peter the Great St. Petersburg Polytechnic University, 2026

ОГЛАВЛЕНИЕ

Предисловие	6
1. Среда разработки	7
2. Внесение данных	9
3. Построение простых графиков	13
4. Линейная аппроксимация	19
5. Оценка погрешностей вычисления коэффициентов линейной модели	25
6. Округление конечных результатов	28
7. Распределения случайных величин	30
8. Гистограммы	33
9. Оценка погрешностей многократных прямых измерений	40
10. Резонансная кривая. Аппроксимация кубическими сплайнами	44
11. Оценка погрешностей косвенных измерений	49
12. Определение добротности по графику резонансной кривой	51
13. Определение добротности по графику кривой затухания	55
Библиографический список	61
Приложения	62
Приложение 1	62
Приложение 2	65
Приложение 3	67
Приложение 4	69

ПРЕДИСЛОВИЕ

Экспериментальные исследования всегда связаны с неточностями, появляющимися как от погрешностей аппаратуры и методов измерения, так и от ошибок экспериментатора. Для повышения достоверности результатов исследования в физике эксперименты проводят многократно, что порождает большие объёмы экспериментальных данных: выборки, временные ряды и др. Однако неавтоматизированные, ручные, расчёты превращают эти большие данные в новый источник ошибок. Поэтому основная цель данного учебного пособия — показать, как существенно снизить вероятность ошибок и трудоёмкость расчётов при обработке результатов лабораторных работ.

Использование компьютера может быть полезно не только для того, чтобы быстро и правильно считать. Оно позволяет каждому студенту применять передовые алгоритмы вычислительной математики для аппроксимации, интерполяции, математического моделирования. Использование компьютера даёт возможность относительно точно оценивать искомые величины между точками измерения, рассчитывать погрешности автоматически, получать статистически эффективные и несмещённые оценки там, где вручную традиционными методами это за разумное время сделать невозможно. В данном учебном пособии значительное внимание уделено построению аппроксимирующих функций и статистическим свойствам оценок коэффициентов моделей.

Ещё одним важным фактором является графическое представление данных эксперимента и результатов их обработки. Качествен-

ное, точное представление многое поясняет читателю, позволяет сразу акцентировать внимание на главных достижениях. При этом оно компактно, легко переносимо. В настоящее время можно получать высококачественные иллюстрации сразу в процессе обработки результатов, без использования промежуточных программ, чему в данном учебном пособии уделено большое внимание.

1. СРЕДА РАЗРАБОТКИ

Работа строится с учетом того, что студенты знакомы с основами языка программирования Python. Если даже они не знакомы с этим, то это не критично, так как сложные алгоритмы не будут рассматриваться. Изложение ведется таким образом, чтобы как можно быстрее написать программу, выдающую реальный результат, который можно применить на занятиях. Далее будут рассмотрены различные частные случаи, первоначальная программа будет усовершенствована.

Вначале следует определить, в какой среде разработки будет вестись работа. Сложно что-то советовать, так как 10 лет, даже пять лет, назад разработчики на Python предпочитали одни среды, теперь — совсем другие. Какие среды разработки будут востребованы в тот момент, когда данное учебное пособие попадёт вам в руки, неизвестно. Поэтому если у вас уже есть какие-то предпочтения, то на них и следует опираться. Если только начинается знакомство с основами языка программирования Python, то можно дать несколько рекомендаций (но нет гарантии, что они будут актуальны в то время, когда вы читаете данное учебное пособие), главное, никогда не пользуйтесь платными средами, для Python всегда можно найти качественные бесплатные среды разработки.

Итак, начнём с *онлайн-редакторов*. На момент написания учебного пособия основная проблема онлайн-редакторов состоит в том, что они не строят графики, а для вас это основное, что требуется от программы. Пока можно рекомендовать **Google Colab**. Бесплатной версии для ваших задач достаточно. С помощью данной программы можно строить графики, есть возможность их скачать и вставить в отчёт по лабораторной работе.

Подходящих *интегрированных сред разработки* (Integrated Development Environment, IDE) существует множество. Вот некоторые из них:

- **IDLE** — редактор, поставляемый вместе с Python. Это базовый, упрощенный режим программирования на Python. Тем не менее IDLE — хороший редактор для начала программирования и понимания основ языка. Есть вариант IDLEX (eXtended — расширенный вариант), в котором есть возможность открытия разных программ во вкладках и нумерация строк слева. При наличии исходного IDLE он может быть установлен через pip — встроенную систему установки модулей Python.

- **Visual Studio Code** — бесплатный редактор кода от Microsoft для Windows, Linux и MacOS. Его возможности — отладка, подсветка синтаксиса, интеллектуальное завершение кода, предопределённые фрагменты кода, рефакторинг и интеграция с Git. Поддерживаются различные языки программирования.

- **PyCharm** — интегрированная среда разработки специально для Python, имеет широкий набор возможностей, таких как автозавершение и инспекции кода, подсветка ошибок, исправления, отладка, система контроля версий и рефакторинг. PyCharm доступна на Microsoft Windows, Linux и MacOS. Бесплатной версии для обучения достаточно.

- **Spyder** — мощная научная интегрированная среда программирования, написанная на Python и для Python. Она разработана учеными, инженерами и аналитиками данных для них. Spyder обладает уникальным сочетанием возможностей. Продвинутое редактирование, анализ, отладка и профилирование сочетаются с возможностями исследования данных, интерактивного выполнения, глубокой инспекции кода и красивой визуализацией. Возможна установка через pip, по крайней мере на Linux, где требуемые системные библиотеки уже, как правило, установлены.

- **JupyterLab** — среда разработки для data science, в которой можно строить графики, анализировать данные и работать с интерактивными блокнотами Jupyter Notebook. JupyterLab поставляется со всеми необходимыми инструментами и библиотеками.

Среда разработки, помимо Python, официально поддерживает Julia и R — популярные языки программирования для анализа данных.

— **PyZO** — программа со смешным названием, но довольно функциональная, до сих пор вполне годная для научного программирования. Это пионер сред для Python3.

Далее оперативно выбираем, устанавливаем понравившуюся среду разработки и приступаем к обработке данных.

2. ВНЕСЕНИЕ ДАННЫХ

В основном при проведении лабораторных работ вам предстоит снимать одномерные данные (например, это выборка значений интервалов времени). Такие данные удобнее всего сохранять в виде одномерных *массивов*. В Python для этого необходимо подключить внешний, не входящий в стандартный дистрибутив, модуль `numpy`. Нужно проверить, что этот модуль установлен, или установить его. Массивы `numpy` как объекты обладают большим набором встроенных удобных математических методов.

Для того чтобы использовать возможности модуля `numpy`, необходимо в самом начале программы подключить этот модуль:

```
#импортируем модуль numpy и называем его далее как np
import numpy as np
```

Именно такой импорт с переименовыванием в настоящее время считается стандартным и общепринятым, он встречается в разных пособиях и на сайтах.

Базовый оператор создания массива называется `array`. Следует создать одномерный массив и вывести, например, его среднее значение на экран:

```
a = np.array([148, 132, 118, 106, 94, 86])
print(a.mean())
```

На экране будет выведено: `114.0`.

Разберём некоторые специфические понятия в этом маленьком коде, которые очевидны тем, кто давно пишет на Python, но могут быть непонятны тем, кто первый раз его видит. Начнём со второй строки. Обратите внимание на круглые скобки после метода `mean`. Они указывают на то, что вызывается и исполняется соответствующая функция. Поскольку Python — язык глубоко объектный, то методы, как и их параметры, результаты, тоже являются объектами. Если забыть поставить скобки, вычисления не будут произведены, а вместо их результатов будут напечатаны строки документации соответствующих функций. То есть `a.mean()` — это результат вычисления — среднее элементов массива `a`, а `a.mean` — это метод вычисления как объект:

```
<built-in method mean of numpy.ndarray object at 0x7cf2165a3210>
```

Большинство стандартных методов работы с массивами (табл. 2.1) имеют аналогичные функции в `numpy`, например вместо метода `a.mean()` можно написать функцию `np.mean(a)`. В своих программах можно использовать любой из этих способов.

Тип результата встроенных функций определяется их природой: `min`, `max` и `sum` всегда того же типа, что и элементы массива, а `mean`, `var` и `std` всегда действительнзначные, поскольку для их вычисления выполняются деление и — для `std` — извлечение квадратного корня. Поэтому, несмотря на то что массив `a` состоял из целых чисел, результат выполнения функции `mean` получился действительный.

Далее вернёмся к первой строчке нашего кода. Может быть непонятно, зачем внутри функции `array`, кроме круглых скобок, нужны ещё и квадратные. Дело в том, что то, что стоит внутри квадратных скобок, в Python называется *списком*. Именно этот список функция `array` преобразует к массиву. Для обработки некоторых лабораторных работ нам будет удобно пользоваться другим способом создания массивов. Если в нашем распоряжении будет иметься *эквидистантный ряд* (**временной ряд**, значения

Таблица 2.1

Методы массивов

№ п/п	Метод	Описание
1	<code>a.sum()</code>	Сумма элементов массива: $\sum_{i=0}^{len(a)-1} a_i = 0.1+0.25-1.75+0.4-0.9 = -1.9$
2	<code>a.mean()</code>	Среднее значение элементов массива: $\bar{a} = \frac{1}{len(a)} \sum_{i=0}^{len(a)-1} a_i = (0.1+0.25-1.75+0.4-0.9) / 5 = -0.38$
3	<code>a.min()</code>	Минимальный элемент массива: <code>-1.75</code>
4	<code>a.max()</code>	Максимальный элемент массива: <code>0.4</code>
5	<code>a.var()</code>	Дисперсия элементов массива (по умолчанию смещённая оценка): $\sigma^2 = \frac{1}{len(a)} \sum_{i=0}^{len(a)-1} (a_i - \bar{a})^2 =$ $= ((0.1+0.38)^2 + (0.25+0.38)^2 + (-1.75+0.38)^2 + (0.4+0.38)^2 + (-0.9+0.38)^2) / 5 =$ $= 3.383 / 5 = 0.6766$
6	<code>a.std()</code>	Среднеквадратичное отклонение (корень из дисперсии): <code>0.822556988907</code>

которого взяты через равные промежутки времени), то удобнее будет использовать функцию `arange`. Данная функция позволяет сразу создать массив-диапазон, причём можно использовать дробный шаг. Функция `arange` имеет три обязательных и один необязательный параметры. Первый параметр — начальное значение массива, второй параметр — конечный элемент массива (не включается в массив), третий параметр — шаг между значениями, четвёртый именованный параметр `dtype` — тип данных массива (всегда будут действительные числа `float`, даже если изначально

по природе данные будут целыми `int`, это необходимо будет при дальнейших вычислениях).

Например, было необходимо делать замеры значения сопротивления металла при нагревании на каждые 3 °C. Тогда ряд температуры будет эквидистантным, а ряд сопротивления — каким получится, скорее всего, неэквидистантным. Тогда есть смысл вносить их в программу в следующем виде:

```
tempr = np.arange(25, 73, 3, dtype=float)
Rm = np.array([99.1, 99.5, 100.0, 101.0, 102.3, 103.5,
104.8, 106.1, 107.0, 108.8, 110.7, 112.4, 114.2, 114.9,
115.7, 116.6])
```

То есть эквидистантный ряд температуры генерируется с помощью функции `arange`, а неэквидистантный ряд сопротивления — вручную с помощью функции `array`.

Кроме `arange` есть ещё функция `linspace`, тоже создающая массив-диапазон. Её первые два аргумента те же, что и у `arange`: начало и конец диапазона, а третий аргумент — количество элементов в диапазоне. Эта функция будет полезна, когда точно известно, сколько значений должно получиться, например, необходимо сделать ровно 50 замеров интервала времени. По умолчанию эти функции по-разному обрабатывают второй аргумент: `arange` берёт его неэквидистантно, `linspace` — эквидистантно.

В случае когда используется функция `arange`, будет полезно на всякий случай проверять, что длины массивов (в данном случае `tempr` и `Rm`) совпадают, иначе далее могут возникнуть проблемы с построением графиков. Для этого используется встроенная функция `len`:

```
print(len(tempr), len(Rm))
```

Вывод программы: 16 16.

Данные внесены. Далее перейдём к построению графиков.

3. ПОСТРОЕНИЕ ПРОСТЫХ ГРАФИКОВ

Построение графиков — один из главных этапов обработки данных. Для построения графиков в Python есть модуль `matplotlib`. Он тоже нестандартный, поэтому может понадобиться его дополнительно установить. Непосредственно за построение графиков отвечает библиотека `pyplot` модуля `matplotlib`. Основная команда для построения графиков — `plot`. Импортируем необходимую библиотеку, не забывая при этом импортировать и модуль `numpy`. Возьмём два массива, рассмотренных ранее, построим график зависимости сопротивления металла от температуры, а также поменяем стиль и цвет линий графика, сделаем заголовок, нанесём сетку, подпишем оси. В итоге получим:

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams['font.size'] = 18.0
tempr = np.arange(25, 73, 3)
Rm = np.array([99.1, 99.5, 100.0, 101.0, 102.3, 103.5,
104.8, 106.1, 107.0, 108.8, 110.7, 112.4, 114.2, 114.9,
115.7, 116.6])
plt.figure(figsize = (14, 10))
plt.plot(tempr, Rm, 'o', color = 'black')
plt.title('Зависимость сопротивления металла от
температуры')
plt.xlabel(r'$t$, °C')
plt.ylabel(r'$R_m$, Ом')
plt.grid()
plt.show()
```

В этом коде присутствует много нового и непонятного. Последовательно всё разберём. Вначале идёт понятный нам импорт модулей и библиотек, в том числе параметра `rcParams['font.size']`, который позволяет указать размер шрифта по умолчанию для всех текстовых элементов графика. Можно внутри каждой

команды, описанной далее, использовать именованный аргумент `fontsize`. Но часто удобнее один раз вначале программы выставить шрифт (нам нужен крупный шрифт, так как затем графики будут вставляться в отчёт по лабораторной работе) и забыть об этом.

Без функции `figure` можно было бы обойтись, если бы графики строились только для просмотра, но нам нужно будет их вставлять в отчёт по лабораторной работе, поэтому размер графика, как и шрифта, должен быть крупным. Поэтому потребуется только один именной аргумент этой функции — `figsize`, который задаёт ширину и высоту в дюймах (14, 10).

Далее разберёмся с синтаксисом команды `plot`. В простейшем случае она требует единственный аргумент — массив. Тогда по оси абсцисс будут отложены номера элементов массива, а по оси ординат — эти элементы. Но часто (в том числе и в данном случае) оказывается необходимо отложить по оси абсцисс не номера точек, а какие-то осознанные значения. Это можно сделать, передав функции `plot` два массива, как это и сделано ранее в программе. Если оставить внутри функции `plot` только два этих параметра, то можно увидеть на графике синюю сплошную ломаную. Чтобы изменить тип линии или заменить её на маркеры, можно указать дополнительный аргумент. Например, строка из одного символа дефиса `'-'` соответствует поведению по умолчанию сплошной линии, `'--'` — прерывистой линии, `'.'` — пунктирной линии (табл. 3.1). При этом маркеры (кружочки, квадраты, ромбы) будут располагаться только в тех местах, где есть данные.

Маркеры и линии можно комбинировать. В программе мы отказались от заполнения пространства между имеющимися данными линией (далее будет понятно, почему так было сделано), а в качестве маркера был выбран кружок `'o'`. В результате на месте расположения данных появятся жирные кружки. Чтобы изменить цвет графика, достаточно указать дополнительный именованный параметр `color`. В данном случае было использовано специальное именованное значение `'black'`, соответствующее чёрному цвету. Для базовых цветов существуют и однобуквенные обозначения: `'b'` — синий; `'g'` — зелёный; `'r'` — красный; `'c'` — голубой; `'m'` —

пурпурный; ‘y’ – жёлтый; ‘k’ – чёрный; ‘w’ – белый. В настоящее время (помните, что к тому моменту, когда вы возьмёте в руки это учебное пособие, всё уже может измениться) в `matplotlib` по умолчанию используется так называемая «категорическая

Таблица 3.1

Тип линий (`linestyle`) или тип маркеров (`markers`)

№ п/п	Строка форматирования	Описание
1	<code>'-'</code>	Сплошная линия
2	<code>'--'</code>	Прерывистая линия
3	<code>'-.'</code>	Штрихпунктирная линия
4	<code>':'</code>	Пунктирная линия
5	<code>'.'</code>	Маркер «точка»
6	<code>','</code>	Маркер «пиксель»
7	<code>'o'</code>	Маркер «кружок»
8	<code>'v'</code>	Маркер «треугольник вниз»
9	<code>'^'</code>	Маркер «треугольник вверх»
10	<code>'<'</code>	Маркер «треугольник влево»
11	<code>'>'</code>	Маркер «треугольник вправо»
12	<code>'s'</code>	Маркер «квадрат»
13	<code>'d'</code>	Маркер «вытянутый ромб»
14	<code>'D'</code>	Маркер «квадратный ромб»
15	<code>'p'</code>	Маркер «пятиугольник»
16	<code>'h'</code>	Маркер «шестиугольник»
17	<code>'*'</code>	Маркер «звёздочка»
18	<code>'+'</code>	Маркер «плюс»
19	<code>'x'</code>	Маркер «крестик»
20	<code>' '</code>	Маркер «вертикальная линия»
21	<code>'_'</code>	Маркер «горизонтальная линия»

палитра T10», состоящая из следующих цветов: ‘tab:blue’, ‘tab:orange’, ‘tab:green’, ‘tab:red’, ‘tab:purple’, ‘tab:brown’, ‘tab:pink’, ‘tab:gray’, ‘tab:olive’, ‘tab:cyan’.

Далее идёт команда для вывода заголовка `title`. В этой программе её аргумент выглядит просто — в виде текстовой строки, затем будем усложнять её вывод. Команды `xlabel` и `ylabel` используются для подписи соответствующих осей. Это кажется интуитивно понятным, а вот то, что написано у них внутри скобок, может вызвать много вопросов.

Часто необходимо, чтобы в подписях на осях, легенде или заголовке содержались верхние или нижние индексы, греческие буквы, различные значки и прочие математические символы. Для этого `matplotlib` имеет режим совместимости с командами LaTeX. Чтобы использовать его, нужно поставить перед строкой символ ‘`r`’, поскольку и в Python, и в LaTeX символ ‘`\`’ является специальным и вводит команды. Строка, начинающаяся с ‘`r`’, называется *сырой*. Внутри сырой строки нужно поставить символ ‘`$`’, с которого начинается и которым заканчивается формула LaTeX. Внутри пары из ‘`$`’ можно использовать многие специальные обозначения (табл. 3.2), например, ‘`_`’ для нижнего индекса, ‘`^`’ — для верхнего. Если в индекс входит несколько символов, следует заключить их в фигурные скобки {}, причём фигурные скобки не отображаются. Всё, что находится внутри знаков ‘`$`’, будет написано курсивом. Единицы измерения принято писать прямым шрифтом, поэтому их нужно вынести за знак ‘`$`’.

В примере использовались только команды верхнего (`r'$t`, `^{\circ}C`) и нижнего индексов (`r'$R_m$`, `кОм`). В итоге на оси абсцисс будет подпись « $t, ^{\circ}C$ », а на оси ординат — « R_m , кОм».

В табл. 3.3 приведены LaTeX’овские обозначения для греческих букв. Далее нам часто будет полезна данная таблица.

Команда `grid` используется для создания сетки — набора горизонтальных и вертикальных линий, которые накладываются на график. Сетка помогает визуально оценивать значения данных и облегчает чтение графика, особенно когда на нём представлено много информации.

Таблица 3.2

Стандартные команды LaTeX для работы с математическими формулами

Сырая строка	Обозначение	Результат
<code>r'\$x_y\$'</code>	Нижний индекс	x_y
<code>r'\$x^y\$'</code>	Верхний индекс	x^y
<code>r'\$\dot{x}\$'</code>	Производная	$\dot{x}$
<code>r'\$\overline{x}\$'</code>	Среднее (черта)	$\overline{x}$
<code>r'\$\widehat{x}\$'</code>	Оценка (крышка)	$\hat{x}$
<code>r'\$\overrightarrow{x}\$'</code>	Стрелка (вектор)	$\overrightarrow{x}$
<code>r'\$\cdot\$'</code>	Скалярное произведение	$\cdot$
<code>r'\$\times\$'</code>	Векторное произведение	$\times$
<code>r'\$\angle x \ \rangle\$'</code>	Угловые скобки	$\langle x \rangle$
<code>r'\$\perp\$'</code>	Перпендикулярно	$\perp$
<code>r'\$\parallel\$'</code>	Параллельно	$\parallel$
<code>r'\$\pm\$'</code>	Плюс/минус	$\pm$
<code>r'\$\rightarrow\$'</code>	Стрелка	$\rightarrow$
<code>r'\$\infty\$'</code>	Бесконечность	∞
<code>r'\$\lim_{n \rightarrow \infty}\$'</code>	Предел	$\lim_{n \rightarrow \infty}$
<code>r'\$\frac{x}{y}\$'</code>	Дробь	$\frac{x}{y}$
<code>r'\$\sqrt[n]{x}\$'</code>	Корень n -й степени	$\sqrt[n]{x}$
<code>r'\$\ln\$'</code>	Натуральный логарифм	$\ln$
<code>r'\$\approx\$'</code>	Примерно равно	$\approx$
<code>r'\$\neq\$'</code>	Знак неравенства	$\neq$
<code>r'\$\leq\$'</code>	Меньше или равно	$\leq$
<code>r'\$\geq\$'</code>	Больше или равно	$\geq$
<code>r'\$\nabla\$'</code>	Оператор набла	∇
<code>r'\$\angle\$'</code>	Знак угла	$\angle$

Греческие буквы в LaTeX

α \alpha	β \beta	γ \gamma	δ \delta
ε \varepsilon	ζ \zeta	η \eta	θ \theta
ι \iota	κ \kappa	λ \lambda	μ \mu
ν \nu	ξ \xi	π \pi	ρ \rho
σ \sigma	τ \tau	υ \upsilon	φ \varphi
χ \chi	ψ \psi	ω \omega	Δ \Delta
Θ \Theta	Λ \Lambda	Ξ \Xi	Σ \Sigma
Υ \Upsilon	Φ \Phi	Ψ \Psi	Ω \Omega

Команды plot, title, xlabel, grid и др. формируют изображение, поэтому они должны предшествовать команде вывода на экран show, порядок относительно друг друга – произвольный.

На рис. 3.1 изображён получившийся график. Стоит обратить внимание на то, что все подписи имеют большой, хорошо читаемый, размер.

Зависимость сопротивления металла от температуры

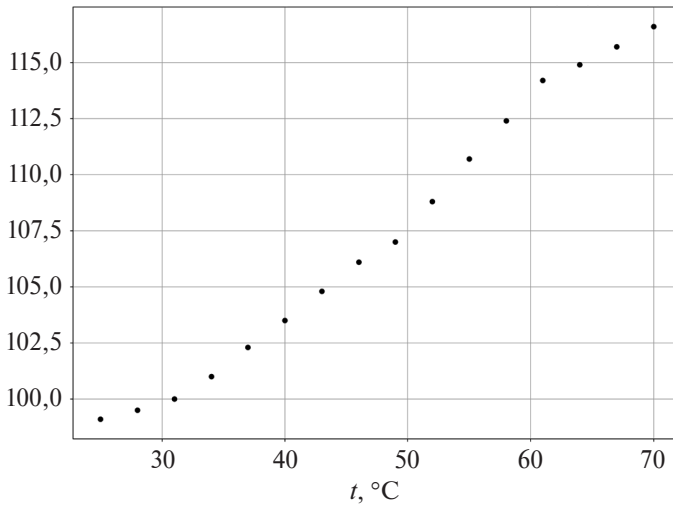


Рис. 3.1. Пример построения графика

После того как вы научились строить графики, следует перейти к следующей задаче, которая ставится примерно в 90 % лабораторных работ по физике (в том числе в той, которая сейчас используется для примера), — аппроксимации получившейся зависимости прямой линией.

4. ЛИНЕЙНАЯ АППРОКСИМАЦИЯ

В широком смысле *аппроксимация* — замена одного объекта другим, в некотором смысле близком к исходному. В каком смысле — решает исследователь в зависимости от того, какие свойства он хочет сохранить. Необходимо помнить, что аппроксимация — это не просто упрощение. Часто целью аппроксимации является замена типа объекта на другой, имеющий дополнительные свойства.

Так, если имеется последовательность пар значений — точек с координатами $\{x_n, y_n\}_{n=0}^{N-1}$, можно заменить её на функцию $y = f(x)$, проходящую через эти точки или в некотором смысле близко к ним, т. е. аппроксимировать набор точек функцией. Функция имеет новые свойства, например, её можно дифференцировать и интегрировать, в то время как к исходному ряду $\{x_n, y_n\}_{n=0}^{N-1}$ такие действия были неприменимы.

При аппроксимации экспериментальных данных с помощью функции можно выделить несколько наиболее распространённых случаев.

— *Интерполяция*. Основная цель аппроксимации состоит в дополнении измеренных данных значениями, лежащими между измеренными. При этом подразумевается, что используемая функция пригодна только для интерполяции и не может описывать данные за пределами области измерения.

— *Экстраполяция*. Основная цель аппроксимации — получение данных за пределами области измерения (например, прогноз), т. е. предполагается, что аппроксимирующая функция,

построенная по имеющимся данным, отражает некоторую общую закономерность и пригодна для экстраполяции.

— *Сжатие данных.* Основная цель аппроксимации — уменьшение объёма данных. То есть нужно заменить большое число измеренных значений небольшим числом коэффициентов аппроксимирующей функции, пусть и с некоторыми потерями.

— *Аппроксимация зашумленных данных.* Основная цель аппроксимации состоит в сглаживании и борьбе с шумами измерений, при этом большой объём экспериментальных данных необходим для того, чтобы увеличить точность определения относительно малого числа коэффициентов модели, явно или неявно проводя усреднение.

Когда в лабораторной работе стоит задача аппроксимации, аппроксимирующая зависимость чаще всего задаётся явно в виде некоторой функции с неизвестными коэффициентами (например, степенного многочлена). В такой постановке задача аппроксимации сводится к задаче нахождения коэффициентов. При этом вы автоматически избавляетесь от шумов измерения, неизбежно возникающих в реальном эксперименте. Практически во всех лабораторных работах задача сведена к аппроксимации имеющихся зависимостей прямой (линейной функцией типа $y = kx + b$) для получения в итоге углового коэффициента k . Это объясняется следующим образом: геометрический смысл углового коэффициента — производная $\frac{dy}{dx}$, которая часто представляет собой иско-

мую физическую величину.

Вернёмся к нашему примеру, в котором по зависимости сопротивления металла от температуры необходимо определить температурный коэффициент электрического сопротивления. Формула для определения температурного коэффициента сопротивления металла следующая:

$$\alpha = \frac{1}{R_0} \cdot \frac{\Delta R_m}{\Delta T},$$

где R_0 — сопротивление металла при температуре $t = 0$ °С. Замеры начинались при комнатной температуре (в аудитории явно было

тепло — 25 °С), следовательно, данных при 0 °С нет. Поэтому будем делать экстраполяцию в область более низких температур. По формуле также видно, что необходимо получить угловой коэффициент $\frac{\Delta R_m}{\Delta T}$.

Для определения коэффициентов аппроксимации линейной функции (k и b) можно использовать аналитические выражения (4.1), (4.2):

$$k = \frac{\sum_{i=0}^{N-1} x_i y_i - \frac{1}{N} \sum_{i=0}^{N-1} x_i \sum_{i=0}^{N-1} y_i}{\sum_{i=0}^{N-1} x_i^2 - \frac{1}{N} \left(\sum_{i=0}^{N-1} x_i \right)^2}; \quad (4.1)$$

$$b = \frac{1}{N} \left(\sum_{i=0}^{N-1} y_i - k \cdot \sum_{i=0}^{N-1} x_i \right). \quad (4.2)$$

Вычисление коэффициентов линейной аппроксимации оформите в виде отдельной функции: так будет удобнее использовать её сразу для нескольких наборов данных в одной программе и копировать её из программы в программу (следует помнить, что почти во всех лабораторных работах требуется линейная аппроксимация):

```
def coeff(x, y):
    k = (sum(x*y) - sum(x) * sum(y) / len(y)) / (sum(x**2) -
        sum(x)**2 / len(y))
    b = 1 / len(y) * (sum(y) - k * sum(x))
    return (k, b)
```

Функции в программировании можно представить как изолированный блок кода, обращение к которому в процессе выполнения программы может быть многократным. В первую очередь такие блоки инструкций нужны, чтобы сократить объем исходного кода: рационально вынести часто повторяющиеся выражения

в отдельный блок и затем по мере надобности обращаться к нему. Правильным считается расположение функций перед телом основной программы. Если одна функция вызывает вторую, то вторую функцию принято писать над первой. В *компилируемых языках программирования* (Fortran, С и его потомки, Pascal и его потомки) иначе просто не работает, если только не используются заголовочные файлы или специальная секция описания заголовков функций в начале программы или модуля. Python, как и MATLAB, PHP, Javascript, являются *интерпретируемыми языками*, и в них функцию можно поместить почти где угодно. Так делать не стоит.

Инструкция `def` — это команда языка программирования Python, позволяющая создавать функцию; `coeff` — это имя нашей функции, которое (как и имена переменных) может быть почти любым, но желательно осмысленным. Затем в скобках перечисляются параметры функции, у нас это массивы x и y . Если параметров нет, то скобки остаются пустыми. Далее идет двоеточие, обозначающее окончание заголовка функции. После заголовка с новой строки и с *отступом* следуют выражения тела функции. Отступ необходим, он определяет уровни вложенности и служит аналогом операторных скобок `{ }` в С. В конце тела функции обычно присутствует инструкция `return`, после которой следует значение или выражение, являющееся результатом работы функции. Именно оно будет подставлено в главной (вызывающей) программе на место функции. Принято говорить, что функция «возвращает значение», в данном случае — функция возвращает коэффициенты аппроксимации k и b . В Python результатом функции может быть только одно значение или выражение. Если необходимо в качестве результата выдать значения сразу нескольких переменных, используют кортеж (ещё один тип данных, неизменяемый список, для его задания используются круглые скобки `( )`).

Попробуем вызвать нашу функцию в основной программе, а заодно сделать аппроксимацию: интерполяцию (дополнить имеющиеся измерения значениями, лежащими, например, через каждый $0,5\text{ }^{\circ}\text{C}$) и экстраполяцию (продолжить прямую в область более низких температур — чуть менее $0\text{ }^{\circ}\text{C}$):

```

k, b = coeff(temp_r, Rm)
x = np.arange(-1, 73, 0.5)
y = k*x + b

```

В первой строчке кода вызывается функция `coeff`, в качестве параметра `x` в функцию передаётся массив `temp_r`, в качестве параметра `y` — массив `Rm`, в переменную `k` записывается нулевой `[0]` элемент кортежа, который возвращает функция `coeff`, а в переменную `b` — первый `[1]` элемент этого же кортежа.

Обратите внимание, что коэффициент `b` представляет собой смещение по оси ординат (сопротивлений) и является не чем иным, как сопротивлением металла при температуре $t = 0\text{ }^{\circ}\text{C}$ (R_0), а коэффициент `k` — это угловой коэффициент, равный $\frac{\Delta R_m}{\Delta T}$. Следовательно, исходный температурный коэффициент сопротивления металла можно вычислить по формуле: $\alpha = \frac{k}{b}$.

Вернёмся к коду. В третьей строке создаётся массив `x` для диапазона температур $[-1, 73)\text{ }^{\circ}\text{C}$ с шагом $0,5\text{ }^{\circ}\text{C}$. В четвёртой строке вычисляются значения сопротивлений металла (массив `y`) с помощью нового массива `x` и коэффициентов аппроксимации `k` и `b`.

Теперь отложим получившуюся зависимость на графике. Аппроксимирующие прямые (и кривые) будем рисовать сплошными линиями, как бы подчёркивая их *непрерывную суть*. Теперь становится ясно, почему исходные данные рисовались точками, — чтобы подчеркнуть их *дискретную* природу. Добавим эту информацию на график в виде легенды. Также добавим в заголовок графика полезную информацию в виде получившейся аппроксимирующей функции и рассчитанного температурного коэффициента сопротивления металла:

```

plt.plot(temp_r, Rm, 'o', color = 'black', label = 'Исходные данные')
plt.plot(x, y, color = 'grey', label = 'Аппрок-
симирующая прямая')

```

```
plt.legend(loc='best')
plt.title('Зависимость сопротивления метал-
ла от температуры\n'+y='+str(round(k,3))+ 'x'+s
tr(round(b,3))+ '\nТемпературный коэффициент со-
противления металла '+r'$\alpha = $'+str(round(k/
b*1000,3))+r'$\cdot 10^{\{-3\}}\sim K^{\{-1\}}$')
```

Чтобы сделать легенду средствами matplotlib, нужно, во-первых, указать при построении каждой кривой требуемую подпись с помощью параметра с ключевым словом `label`, во-вторых, вызвать специальную функцию `legend`, рисующую легенду. У функции `legend` есть параметр с именем `loc`, отвечающий за расположение. Параметр `loc` может принимать числовые значения с нуля до 10 либо соответствующие им строковые

Зависимость сопротивления металла от температуры $y = 0,429x + 86,904$
Температурный коэффициент сопротивления металла $\alpha = 4,938 \cdot 10^{-3} \text{ K}^{-1}$

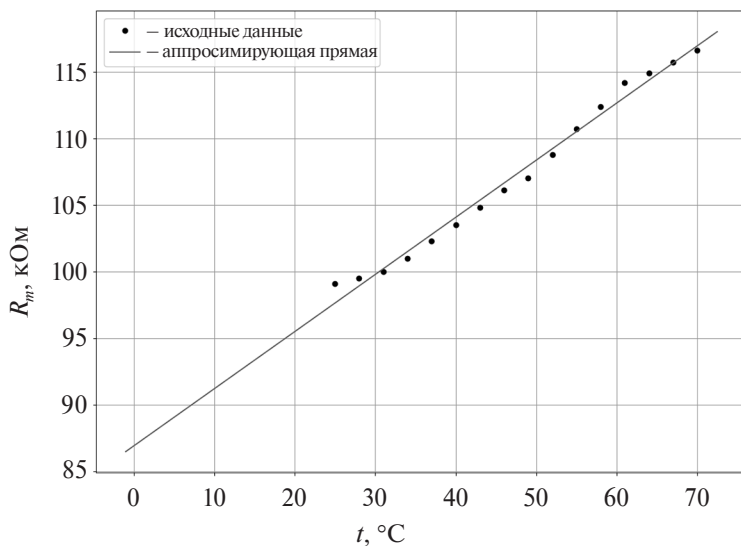


Рис. 4.1. График, содержащий исходные данные и аппроксимирующую прямую

значения типа 'upper right' — верхний правый угол или 'lower center' — внизу посередине. Значение 0 соответствует строке 'best' — Python сам пытается выбрать такое положение легенды, чтобы она не заслоняла график.

Внутри функции `title` использовались до этого неизвестные нам обозначения и функции: `'\n'` — символ перевода строки (печати текста с новой строки, т. е. с левого края на строке ниже, он же Enter); функция `'str'` — функция преобразования числа в строку; функция `'round'` — функция округления до обозначенного знака после запятой (в данном случае — до третьего знака).

Итак, вот что у нас в итоге получилось (рис. 4.1). Посмотрев на график, можно легко ответить, что угловой коэффициент $k = 0,429$, сопротивление $R_0 = 86,904$ кОм, а температурный коэффициент сопротивления металла $\alpha = 4,938 \cdot 10^{-3} \text{ K}^{-1}$.

На этом сравнительно лёгкая и очевидная часть обработки экспериментальных данных закончилась, в большинстве случаев на этом можно остановиться. Далее будем рассматривать вопросы оценки погрешностей.

5. ОЦЕНКА ПОГРЕШНОСТЕЙ ВЫЧИСЛЕНИЯ КОЭФФИЦИЕНТОВ ЛИНЕЙНОЙ МОДЕЛИ

Для оценки значения физической величины можно использовать не просто некоторое полученное фиксированное число $x_{\text{прак}}$ (например, рассчитанный ранее температурный коэффициент сопротивления металла $\alpha_{\text{прак}}$), а интервал значений, в пределах которого может лежать её истинное значение. Если распределение погрешности симметрично (будем исходить из этой нулевой гипотезы), этот интервал может быть записан как $x = x_{\text{прак}} \pm \Delta x$, где Δx — *абсолютная погрешность*. Эта запись означает, что исследуемая величина лежит в *доверительном интервале* $x \in (x_{\text{прак}} - \Delta x; x_{\text{прак}} + \Delta x)$ с некоторой вероятностью. Рассмотрим, какова эта вероятность.

Доверительная вероятность α — это вероятность того, что истинное значение измеряемой величины лежит внутри доверительного

интервала. Обычно доверительная вероятность выражается как $\alpha = 1 - p$, где p – *уровень значимости* (в англоязычной литературе часто называемое p -value). При этом отношение полуширины доверительного интервала Δx к среднеквадратичному отклонению σ_x обозначают как $\varepsilon = \frac{\Delta x}{\sigma_x}$.

В физических измерениях, как правило, используется доверительная вероятность $\alpha = 0,68$. В этом случае погрешность считается равной одному *среднеквадратичному отклонению* $\Delta x = \sigma_x$. Такую запись нужно читать так: «Полученное значение лежит в доверительном интервале $x \in (x_{\text{прак}} - \Delta x; x_{\text{прак}} + \Delta x)$ с вероятностью 68 %». В технических и научных исследованиях чаще используется доверительная вероятность $\alpha = 0,95$. В этом случае погрешность считается равной удвоенному среднеквадратичному отклонению $\Delta x = 2\sigma_x$. Во избежание разночтений доверительную вероятность следует указывать отдельно. Другие популярные значения доверительной вероятности приведены в табл. 5.1.

Вернёмся к нашей задаче с расчётом температурного коэффициента сопротивления металла. Рассчитаем оценку дисперсии коэффициентов аппроксимации. Вначале найдём *невязки*, которые также называются остатками модели $\varepsilon_i = y_i - f(x_i, \vec{c})$, где $\vec{c}$ – вектор

Таблица 5.1

**Соответствие доверительных интервалов,
доверительных вероятностей и уровней значимости**

Доверительный интервал $x \pm \varepsilon \sigma_x$	Доверительная вероятность α	Уровень значимости p
$x \pm \sigma_x$	$\approx 68 \%$	0,32
$x \pm 1,645\sigma_x$	$\approx 90 \%$	0,1
$x \pm 2\sigma_x$	$\approx 95 \%$	0,05
$x \pm 2,57\sigma_x$	$\approx 99 \%$	0,01
$x \pm 3\sigma_x$	99,7 %	0,003
$x \pm 3,29\sigma_x$	99,9 %	0,001

коэффициентов аппроксимации, в нашем случае он состоит из двух значений — (b, k) . Затем найдём оценку дисперсии ряда невязок σ_ε^2 по формуле (5.1) с помощью функции `var` (табл. 2.1) из модуля `numpy`. Её именованный аргумент `ddof` позволяет внести поправку, связанную с тем, что невязки — не совсем случайные числа, а получены в результате подгонки нескольких параметров, при этом сумма квадратов невязок делится на длину ряда минус число параметров, указанное в аргументе `ddof`. Полученная таким образом оценка оказывается несмещённой — её среднее совпадает с теоретическим математическим ожиданием. В данном случае было указано `ddof=2`, поскольку использовали линейную функцию с двумя параметрами: b и k , т. е. в этом случае сумма квадратов невязок делилась на $N-2$, а не на N , как это предполагается по умолчанию (5.1):

$$\sigma^2 = \frac{1}{N-2} \sum_{i=0}^{N-1} \varepsilon_i^2. \quad (5.1)$$

Наконец, оценим погрешности коэффициентов аппроксимации (5.2), (5.3):

$$\sigma_k^2 = \frac{\sigma_\varepsilon^2}{\sum_{i=0}^{N-1} (x_i - \bar{x})^2}; \quad (5.2)$$

$$\sigma_b^2 = \frac{\sigma_\varepsilon^2 \sum_{i=0}^{N-1} x_i^2}{N \sum_{i=0}^{N-1} (x_i - \bar{x})^2}. \quad (5.3)$$

В программе все эти вычисления оформите в виде двух функций: для оценки дисперсии ряда невязок `approx_error` и для оценки погрешности коэффициентов аппроксимации `coeff_error`:

```
def approx_error(x, y):
    k, b = coeff(x, y)
    e = y - (k*x+b)
    return np.var(e, ddof=2)
```

```

def coeff_error(x,y):
    ke      =      (approx_error(x,y)/sum((x-np.
mean(x))**2))**0.5
    be      =      ((approx_error(x,y)*sum(x**2))/
(len(x)*sum((x-np.mean(x))**2))**0.5
    return(ke,be)

```

Таким образом, были получены абсолютные погрешности коэффициентов линейной аппроксимации: $\Delta k = \sigma_k$ и $\Delta b = \sigma_b$. Далее можно вычислить абсолютную погрешность искомого коэффициента сопротивления металла $\Delta \alpha$ (5.4):

$$\Delta \alpha = \alpha \sqrt{\left(\frac{\Delta k}{k}\right)^2 + \left(\frac{\Delta b}{b}\right)^2}. \quad (5.4)$$

Программа будет выводить значение косвенного измерения α и его абсолютную погрешность $\Delta \alpha$ пока без округления:

```

a = k/b
dk, db = coeff_error(temp_r, R_m)
da = a * ((dk/k)**2 + (db/b)**2)**0.5
print(a)
print(da)

```

На экран будет выведено:

```
0.004937811997630824 0.00015188057851053214
```

На этом можно остановиться и сделать округление уже в отчёте по лабораторной работе. Можно пойти далее и написать функцию для автоматического округления результатов.

6. ОКРУГЛЕНИЕ КОНЕЧНЫХ РЕЗУЛЬТАТОВ

Материал данной главы относится только к представлению конечных результатов работы. При проведении измерений и дальнейшей их обработке округления проводить не нужно.

Все цифры, начиная с первой ненулевой, называются значащими. Округление результатов измерений нужно начинать с погрешности. Окончательно в абсолютной погрешности необходимо оставлять одну или две значащие цифры. Две цифры сохраняют, если в старшем разряде стоит цифра «2» или «1» (в этом случае отбрасывание второго знака приводит к очень большой ошибке (до 30–50 %), что недопустимо). Если в старшем разряде цифра больше «2», то оставляют одну значащую цифру. После того как погрешность округлена, округляют конечный результат до младшего разряда погрешности (уже по обычным правилам округления).

Напишем отдельную функцию для округления погрешностей и результатов. Пока выведем результаты её работы на экран, но далее такую форму записи можно будет вынести в заголовок графика:

```
def roundoff(a_original, da_original):
    da = da_original
    n = 0
    if abs(da)>=10:
        while abs(da)>=10:
            da = da/10
            if da>=3:
                n = n - 1
    elif abs(da)<1:
        while abs(da)<1:
            da = da*10
            n = n + 1
            if (da>=3) and (da<10):
                break
            elif (da>=1) and (da<3):
                n = n + 1
    else:
        if int(da)>=3:
            n = 0
```

```

else:
    n = n + 1
return (round(a_original,n), round(da_
original,n))

```

Внутри основной программы добавим строчку вызова функции roundoff:

```

print (str(roundoff(a, da)[0]) + ' \
u00B1'+str(roundoff(a, da)[1]))

```

На экран будет выведено: 0.00494 ± 0.00015 .

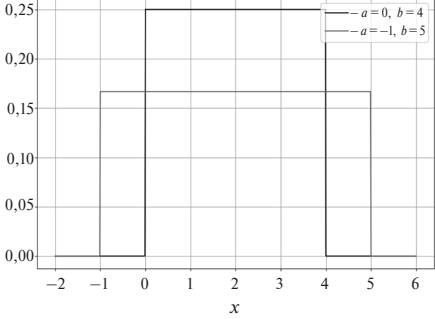
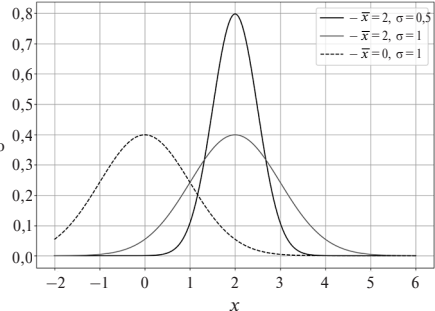
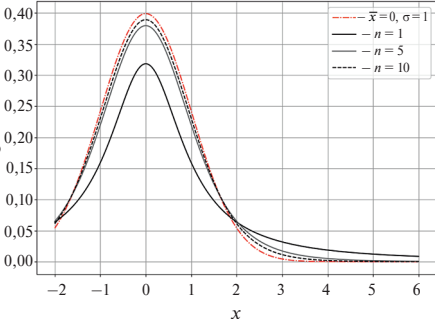
В окончательных результатах можно будет написать $\alpha_{\text{прак}} = (4,94 \pm 0,15) \cdot 10^{-3} \text{ K}^{-1}$ при $\alpha = 0,68$. Если, например, известно, что этот металл — платина, то можно сравнить полученное значение температурного коэффициента сопротивления с табличным $\alpha_{\text{теор}} = 3,93 \cdot 10^{-3} \text{ K}^{-1}$. Тогда в выводах можно написать, что «теоретическое значение температурного коэффициента сопротивления платины *не попадает* в 68%-ный доверительный интервал экспериментально полученного значения». На самом деле легко увидеть (следует посмотреть на данные табл. 5.1 и умножить на соответствующее число полученную ширину полуинтервала $0,15 \cdot 10^{-3}$), что табличное значение не попадёт даже в 99,9%-ный доверительный интервал.

7. РАСПРЕДЕЛЕНИЯ СЛУЧАЙНЫХ ВЕЛИЧИН

Распределение случайной величины — это закон, который каждому значению случайной величины ставит в соответствие вероятность принятия ею этого значения. Один из способов задания распределения случайной величины — *функция плотности распределения* $\rho(x)$. Рассмотрим пять видов непрерывных распределений из теории вероятностей: равномерное, нормальное (Гаусса), Стюдента, экспоненциальное, χ^2 (хи-квадрат) и ещё два из статистической физики: Максвелла и Больцмана (табл. 7.1).

Таблица 7.1

Непрерывные распределения случайных величин

№ п/п	Функция плотности распределения	График
1	Равномерное непрерывное распределение: $\rho(x) = \begin{cases} \frac{1}{b-a}, & x \in [a, b], \\ 0, & x \notin [a, b], \end{cases}$ где a и b – границы отрезка	
2	Нормальное распределение Гаусса: $\rho(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x-\bar{x})^2}{2\sigma^2}\right),$ где $\bar{x}$ – математическое ожидание (среднее значение); σ – среднее квадратичное отклонение	
3	Распределение Стьюдента: $\rho(x) = \frac{\Gamma\left(\frac{n+1}{2}\right)}{\sqrt{n\pi}\Gamma\left(\frac{n}{2}\right)} \left(1 + \frac{x^2}{n}\right)^{-\frac{n+1}{2}},$ где Γ – гамма-функция; n – количество степеней свободы	

№ п/п	Функция плотности распределения	График
4	Экспоненциальное распределение: $\rho(x) = \lambda e^{-\lambda x},$ где λ – средняя скорость, с которой происходят события, показывает, насколько часто события возникают в единицу времени	
5	Распределение χ^2: $\rho(x) = \frac{(1/2)^{n/2}}{\Gamma(n/2)} x^{n/2-1} e^{-x/2},$ где Γ – гамма-функция; n – количество степеней свободы	
6	Распределение Максвелла: $\rho(v) = 4\pi v^2 \left(\frac{m}{2\pi k T} \right)^{3/2} \exp\left(-\frac{mv^2}{2kT} \right),$ где v – модуль скорости молекулы; T – температура системы; m – масса молекулы $m = \frac{M}{N_a}$; M – молярная масса; N_a – число Авогадро; k – постоянная Больцмана	

№ п/п	Функция плотности распределения	График
7	Распределение Больцмана: $n = n_0 \exp\left(-\frac{mgh}{kT}\right),$ где h – высота; n – концентрация молекул газа на высоте h; n_0 – концентрация молекул газа на высоте $h_0 = 0$	

8. ГИСТОГРАММЫ

В гл. 7 были рассмотрены аналитические функции распределения случайных величин. На практике будем иметь набор экспериментальных данных, для которых аналитических формул не существует, но есть другие способы представления.

Гистограмма – это столбчатый график, построенный по экспериментальным данным. Значения разбиваются на ряд интервалов (*бинов*), расположенных в порядке возрастания. Число данных (частота значений), попадающих в каждый интервал, определяет высоту столбцов.

Важно отметить, что говорить о плотности вероятности в контексте гистограммы неправильно, так как гистограммирование трансформирует распределение любого вида в дискретное (рассматривается событие попадания величины в некоторый интервал, число которых счётно), а для дискретной случайной величины не существует функции плотности распределения вероятностей. Тем не менее по виду гистограммы можно составить представление о плотности распределения вероятностей и сделать предположение о типе распределения.

В данной главе рассмотрим пример с многократным измерением заданного промежутка времени. Один из участников эксперимента

фиксирует с помощью механических часов начало и конец промежутка времени (пусть это будет 5 с), подавая напарнику команды на пуск и остановку цифрового секундомера. Второй участник фиксирует значение, которое он видит на секундомере (предполагаем, что с точностью до второго знака после запятой, т. е. приборная погрешность $\Delta t_{\text{приб}} = 0,01$). Таким способом необходимо получить 50 измерений. В этом случае будет необходимо в программе внести вручную в список 50 значений. Для примера можно взять результаты замеров времени одной из студенток прошлых лет:

```
time = np.array([5.07, 5.30, 5.06, 4.92, 5.38,
4.74, 4.87, 4.92, 5.01, 4.86, 4.57, 5.07, 5.00,
5.52, 5.29, 4.88, 5.06, 4.74, 5.07, 5.38, 4.80,
4.95, 4.73, 5.21, 5.24, 4.62, 5.11, 4.63, 4.97,
4.40, 4.87, 4.93, 4.88, 4.99, 4.90, 5.07, 5.36,
5.10, 4.47, 5.30, 4.74, 4.69, 5.35, 5.32, 4.81,
4.88, 4.45, 4.76, 5.07, 5.32])
N = len(time)
```

Также для первоначального написания и последующей отладки программы можно вначале сгенерировать массив нормально распределённых псевдослучайных чисел, используя для этого функцию `normal` из библиотеки `random` модуля `numpy`:

```
N = 50
time = np.random.normal(5, 0.5, N)
```

Функция `normal` на вход получает последовательно три параметра: среднее значение нормального распределения $\bar{t} = 5$; среднеквадратичное отклонение (СКО) распределения $\sigma_t = 0,5$; количество значений в выборке $N = 50$.

Вначале построим гистограмму плотности относительной частоты попадания результатов измерений в заданный интервал

(бин). Количество бинов должно быть целым, близким к $\sqrt{N}$, где N – количество измерений. Так как есть 50 измерений, то будем разбивать весь диапазон от $t_{\min}$ до $t_{\max}$ на семь бинов (следует проверить в программе, что строчка `nbins = int(N**0.5)` выдаёт нужное количество бинов). Воспользуемся функцией `hist` из библиотеки `pyplot`:

```
nbins = int(N**0.5)
plt.hist(time,      bins=nbins,      color='white',
edgecolor='black', density=True)
```

На вход функция `hist` принимает массив входных данных `time`; именованные параметры: `bins` – количество столбцов в гистограмме, `color` – цвет или последовательность цветов, используемых для столбцов в гистограмме, `edgecolor` – цвет границ столбцов гистограммы, `density` – логический параметр (если установлено значение `True`, то гистограмма получается нормализованной – площадь под гистограммой равна 1, частоты преобразуются в плотность вероятности; по умолчанию – `False`).

Это простой способ построения гистограммы: в программе задаётся количество бинов, и она определяет ширину каждого столбца. Есть вариант сложнее, когда ширину столбца задаёт пользователь. Ширина столбца определяется как $\Delta t = (t_{\max} - t_{\min}) / \sqrt{N}$ с округлением строго вверх до n -го знака после запятой (у нас $N = 50$, $n = 2$). Так как округление в данном случае довольно специфическое, то потребуется модуль `decimal`:

```
from decimal import Decimal, ROUND_UP
delta_t      =      float(Decimal((np.max(time)      -
np.min(time)) / nbins). quantize(Decimal(str(1e-2))),
rounding=ROUND_UP)
```

Метод `quantize` из модуля `decimal` позволяет округлить число до определённого количества знаков после запятой или до конкретного показателя степени (в данном случае это 10^{-2} , на что

указывает строка `(str(1e-2))`). `ROUND_UP` — режим округления в модуле `decimal`, который всегда округляет число в большую сторону. Далее построим гистограмму, используя рассчитанную ширину бина. Тогда по оси абсцисс будет отложено время (t , с), а по оси ординат — плотность относительной частоты попадания результатов измерений в выбранный интервал $\frac{\Delta N}{N\Delta t}$ (оценка плотности распределения), где ΔN — количество результатов измерений, попавших в конкретный бин (интервал $[t, t + \Delta t]$):

```
plt.hist(time, bins=np.arange(np.min(time),
np.max(time) + delta_t, delta_t), color='white',
edgecolor='black', density=True)
```

При таком способе построения (за основу берётся минимальное значение в выборке, к нему прибавляется ширина бина, далее с этим шагом увеличиваются начала бинов) получается такой график, как на рис. 8.1, *а*. Правильнее будет найти середину распределения и бины отстраивать от неё. Тогда для тех же данных получится такая гистограмма, как на рис. 8.1, *б*.

```
#середина распределения
medt = (np.max(time) + np.min(time))/2
#полуширина распределения
poldiap = (np.max(time) - np.min(time))/2
plt.hist(time, bins=np.linspace(medt-poldiap,
medt+poldiap, nbins+1), color='white',
edgecolor='black', density=True)
```

Распределение измеренных значений t в полученной выборке можно наглядно сравнить с *нормальным распределением Гаусса*, исходя из того, что функция распределения плотности вероятности есть предел (8.1):

$$\rho(t) = \lim_{\Delta t \rightarrow 0} \frac{\Delta N}{N\Delta t}. \quad (8.1)$$

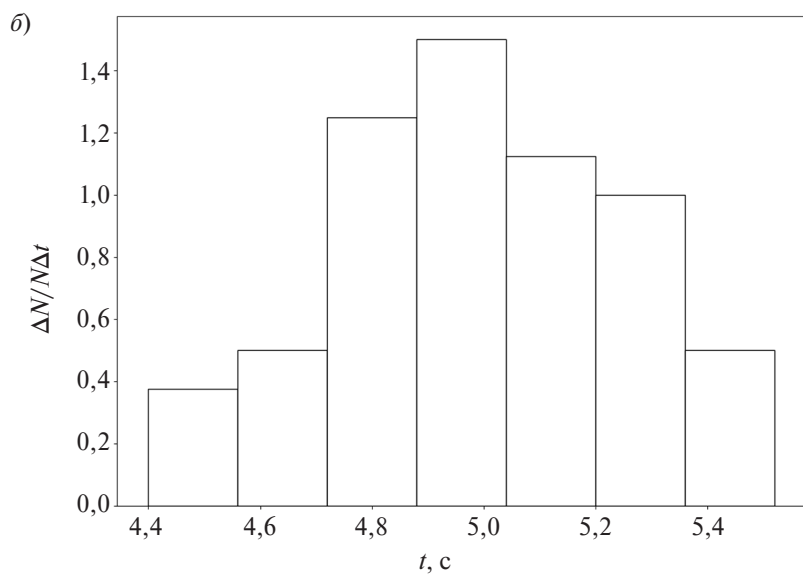
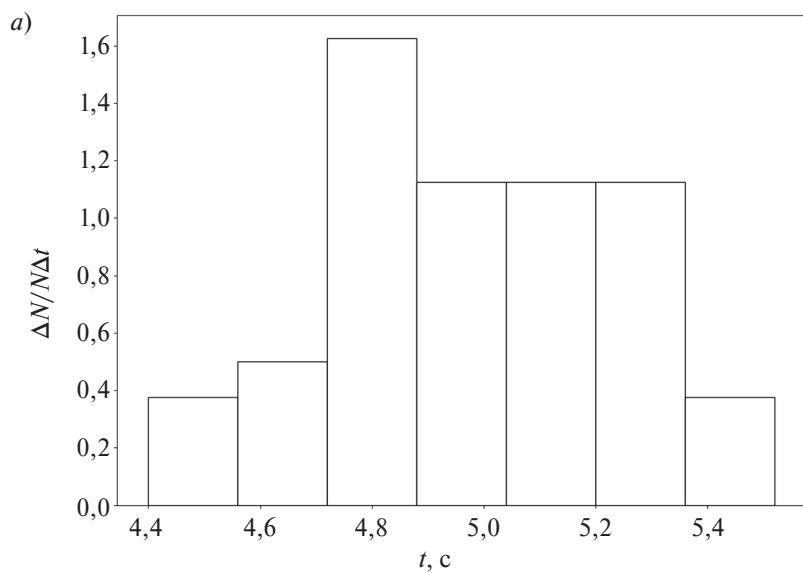


Рис. 8.1. Гистограмма: а – с нецентрированными бинами;
б – с центрированными бинами

Для этого поверх гистограммы построим функцию нормально-го распределения с параметрами (среднее и СКО), как в исходной выборке. Воспользуемся формулой (8.2):

$$\rho(t) = \frac{1}{\sigma_t \sqrt{2\pi}} \exp \left(-\frac{(t - \langle t \rangle)^2}{2\sigma_t^2} \right). \quad (8.2)$$

В программе потребуется вычислить *выборочное среднее* (обозначают угловыми скобками $\langle t \rangle$) по формуле (8.3), *несмещённую оценку* среднеквадратичного отклонения (СКО) по формуле (8.4), создать новый массив времени с более частыми значениями (фактически сделать интерполяцию). Несмещённая оценка СКО имеет смысл в первую очередь при *малых выборках* ($N \leq 50$). Эта поправка, в частности, учитывает, что при одной экспериментальной точке ($N = 1$) *выборочная дисперсия* (σ_t^2) не определена:

$$\langle t \rangle = \frac{1}{N} \sum_{i=0}^{N-1} t_i; \quad (8.3)$$

$$\sigma_t = \sqrt{\frac{\sum_{i=0}^{N-1} (t_i - \langle t \rangle)^2}{N-1}}. \quad (8.4)$$

При $N \rightarrow \infty$ выборочное среднее $\langle t \rangle$ стремится к *математическому ожиданию* случайной величины (обозначают чертой $\bar{t}$). В программе эти вычисления будут выглядеть следующим образом:

```
tm = np.mean(time)
ts = np.std(time, ddof=1)
dt = 0.01
tt = np.arange(np.min(time), np.max(time)+dt, dt)
rho = (np.exp(-1*(tt-tm)**2/(2*ts**2))) / ((2*np.
pi)**0.5*ts)
plt.plot(tt, rho, color = 'grey')
```

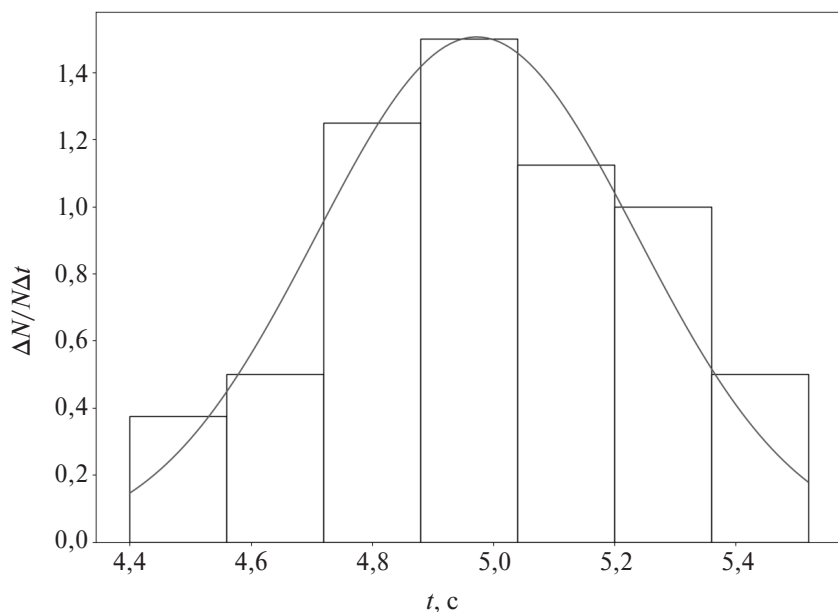


Рис. 8.2. Гистограмма и функция плотности распределения случайной величины t

На рис. 8.2 видно, что гистограмма и кривая функции распределения хорошо соответствуют друг другу: положение максимума кривой попадает в максимальный по высоте бин, число максимумов совпадает, значения максимума гистограммы и максимума функции распределения близки. Это идеальный случай, которого в реальности сложно достичь. На вашей гистограмме может получиться два максимума. Максимум гистограммы может оказаться смещён влево (заранее нажимали «Стоп» на секундомере) или вправо (более типичный случай, нажимали на секундомер с запаздыванием) относительно максимума функции распределения. Иногда гистограмма получается похожей на равномерное распределение; $\rho_{\max}$ может оказаться гораздо выше максимума гистограммы (тогда и гистограмма будет смотреться ниже и шире) или, наоборот, ниже максимума гистограммы (тогда гистограмма будет выше и уже).

9. ОЦЕНКА ПОГРЕШНОСТЕЙ МНОГОКРАТНЫХ ПРЯМЫХ ИЗМЕРЕНИЙ

Далее займёмся расчётом случайной погрешности *многократных прямых измерений*. Дело в том, что выборочное среднее значение $\langle t \rangle$ так же, как и результат отдельного измерения, является случайной величиной. Определение случайной выборки значений $\langle t \rangle$, т. е. проведение не одной серии из N измерений величины t , а набора таких серий, возможно, но не имеет большого смысла, так как теория вероятностей позволяет вычислить оценку среднеквадратичного отклонения выборочного среднего значения по формуле (9.1):

$$\sigma_{\langle t \rangle} = \sqrt{\frac{\sum_{i=0}^{N-1} (t_i - \langle t \rangle)^2}{N(N-1)}} = \frac{\sigma_t}{\sqrt{N}}. \quad (9.1)$$

Погрешность среднего значения $\langle t \rangle$ по результатам N независимых измерений в $\sqrt{N}$ раз меньше погрешности отдельного измерения. Именно этот факт позволяет уменьшать случайные погрешности эксперимента за счёт многократного повторения измерений.

Подчеркнём различия между формулами (8.4) и (9.1):

– величина σ_t характеризует разброс значений в выборке; при нормальном распределении примерно 68 % измерений попадают в интервал $\langle t \rangle \pm \sigma_t$;

– величина $\sigma_{\langle t \rangle}$ характеризует точность, с которой определено среднее выборочное значение $\langle t \rangle$ относительно предельного истинного среднего $\bar{t}$; при этом искомое значение $\bar{t}$ лежит в интервале $\langle t \rangle - \sigma_{\langle t \rangle} < \bar{t} < \langle t \rangle + \sigma_{\langle t \rangle}$ с доверительной вероятностью $\alpha = 68 \%$.

Если расчёты были проведены по данным *большой выборки* ($N > 50$), то отклонение $\bar{t} - \langle t \rangle$ имеет практически нормальное распределение, можно сразу в окончательные результаты написать: « $\bar{t} = (\langle t \rangle \pm \sigma_{\langle t \rangle})$ при $\alpha = 0,68$ » или « $\bar{t} = (\langle t \rangle \pm 3\sigma_{\langle t \rangle})$ при $\alpha = 0,997$ ».

Но если среднее значение $\langle t \rangle$ рассчитывается по данным *малой выборки* (как будет во всех физических экспериментах), то

отклонение $\bar{t} - \langle t \rangle$ имеет распределение Стюдента (оно же t -распределение). Распределение Стюдента сходится к нормальному распределению Гаусса при количестве степеней свободы $n \rightarrow \infty$ (см. табл. 7.1). Но при малых n распределение Стюдента заметно отличается от распределения Гаусса: концентрация отклонений в центральной части распределения у него меньше, а в хвостах — больше (такое распределение также называется «распределение с тяжёлыми хвостами»).

Если случайная величина x_1 распределена по нормальному закону, а случайная величина x_2 — по закону хи-квадрат с n степенями свободы (обычно это сумма квадратов невязок), тогда случайная величина, получаемая как $x_3 = \frac{x_1}{\sqrt{x_2/n}}$ (в знаменателе стоит корень из среднего квадрата невязок, т. е. среднеквадратичное), имеет распределение Стюдента.

По определению *числом степеней свободы* в статистике называют число взаимно независимых элементов информации, используемых для вычисления стандартной ошибки. Число степеней свободы равно числу элементов выборки, из которого вычтено число условий, связывающих данные, т. е. у нас $n = N - 1$.

На практике часто решается задача о том, можно ли утверждать, что некоторое измеренное или рассчитанное значение принадлежит некоторому (нормальному) распределению (часто теоретическому). В первую очередь проверяют гипотезу о том, что среднее для измеренной величины неотлично от среднего для этого теоретического распределения. Такую проверку называют *t -местом Стюдента*, поскольку для проверки эквивалентности средних используется t -распределение (ошибка среднего, как было отмечено ранее, в общем случае подчиняется именно этому распределению). При этом к словам « t -тест Стюдента» иногда добавляют «для средних значений», поскольку также существует тест для дисперсий (нормальное распределение, напомним, определяется двумя параметрами: средним и дисперсией, значит, равенство этих двух параметров означает эквивалентность самих распределений).

Бывают *односторонние* и *двусторонние* t -тесты. Разница между ними заключается в том, что необходимо понять из эксперимента. Если есть уверенность, что отклонение от среднего возможно только в одну сторону следует использовать односторонний тест. Но, как правило, придется опираться на двусторонний тест, потому что бывает неизвестно, в какую сторону отклонится среднее. Разница состоит в том, что при уровне значимости $p = 0,05$ в одностороннем тесте вся критическая зона будет собрана в одном из хвостов распределения (если проверяется гипотеза, что выборочное среднее выше известного, то критическая зона будет собрана в правом хвосте, если ниже — то в левом хвосте). В случае двустороннего теста эта критическая область ($p = 0,05$) обозначится с обеих сторон распределения (по $0,025$ с каждой).

В случае малой выборки доверительный интервал оценки среднего рассчитывается следующим образом (9.2):

$$\langle t \rangle - t_{\alpha, n} \cdot \sigma_{\langle t \rangle} < \bar{t} < \langle t \rangle + t_{\alpha, n} \cdot \sigma_{\langle t \rangle}, \quad (9.2)$$

где $t_{\alpha, n}$ — коэффициент Стьюдента (квантиль распределения Стьюдента, критическое значение распределения Стьюдента) для доверительной вероятности α и числа степеней свободы n .

Ранее, когда не было Python и мгновенных вычислений, проверка гипотез базировалась на вычислениях *критических значений распределения*. Рассчитанные значения заносились в таблицы, которые потом использовались для оценки экспериментов. Критическое значение — это квантиль t -распределения или такое число, что вероятность попасть правее него равна $p = 1 - \alpha$ (или $p/2$ при двухстороннем тесте). Чтобы найти критические значения t -распределения при разных степенях свободы и уровнях значимости, приходилось решать интегральное уравнение:

$$\int_{t_{\alpha, n}}^{\infty} \rho_T(t) dt = \frac{p}{2},$$

т. е. подбирать такое значение $t_{\alpha, n}$, при котором площадь хвоста t -распределения равна заданной доверительной вероятности. Это делалось численными методами, результаты собирались в таблицы.

В настоящее время всё значительно проще: во-первых, эти таблицы давно существуют, во-вторых, критическое значение можно получить напрямую программно (например, с использованием метода `ppf` (percent point function) функции `t` из библиотеки `stats` модуля `scipy`). Импортируем в программу нужную функцию, пропишем требуемые доверительную вероятность и уровень значимости, рассчитаем соответствующий коэффициент Стьюдента. Для проверки можно вначале его значение просто вывести на экран:

```
from scipy.stats import t
a = 0.68 #доверительная вероятность
p = 1-a #уровень значимости
tan = t.ppf(1-p/2, N-1)
print(tan)
```

Если тест односторонний, т. е. существует уверенность в том, в какую сторону возможно отклонение, то можно подставлять $1-p$ вместо $1-p/2$.

Усовершенствуем данную программу, добавив в заголовок рассчитанные ранее величины:

```
plt.title(r'$\Delta t = $' + str(delta_t) + r'$,
\sigma_t = $' + str(round(ts,3)) + r'$,
\sigma_{\langle t \rangle} = $' +
str(round(tst,3))+r'$, \rho_{\max} = $' +
str(round(0.4/ts,3)) + r'$, \overline{t}
= ($' + str(roundoff(tm,tst*tan)[0]) +
r'$\pm$' + str(roundoff(tm,tst*tan)[1]) + `
при ` + r'$\alpha = $' + str(a))
```

В итоге получим график (рис. 9.1), на котором в заголовке выведены все искомые величины: ширина бина Δt ; несмещённая оценка среднеквадратичного отклонения σ_t ; оценка среднеквадратичного отклонения выборочного среднего значения $\sigma_{\langle t \rangle}$; максимум

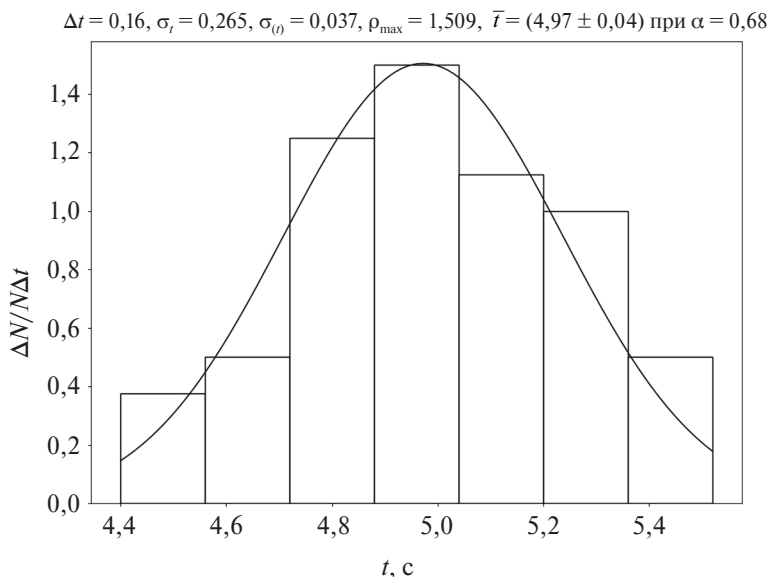


Рис. 9.1. Итоговый график, в заголовке которого выведены все искомые величины

функции $\rho(t)$, равный $\rho_{\max} \approx \frac{0,4}{\sigma_t}$; среднее значение выборки с погрешностью $\bar{t} = \langle t \rangle \pm t_{\alpha,n} \cdot \sigma_{\langle t \rangle}$ при определённой доверительной вероятности и с округлением до нужного порядка (для этого в самый верх вашей программы нужно добавить функцию `roundoff`, описанную в гл. 6).

10. РЕЗОНАНСНАЯ КРИВАЯ. АППРОКСИМАЦИЯ КУБИЧЕСКИМИ СПЛАЙНАМИ

Нередко в лабораторных работах ставится цель снятия резонансной кривой в разных физических задачах: изгибных колебаний стального стержня; акустической камеры; колебаний поршня, сжимающего столб воздуха; колебательных *LC*- и *RLC*-контуров.

Вначале введём несколько терминов, чтобы говорить на понятном друг другу языке. *Резонанс* — явление, при котором колебательная система начинает колебаться с максимальной амплитудой под действием внешней периодической силы, частота которой совпадает или близка к собственной частоте системы. Колебания, которые происходят под воздействием внешней периодически изменяющейся силы, называются *вынужденными*. Колебания, возникающие в системе под действием внутренних сил после того, как система была выведена из состояния равновесия, называются *свободными*. В реальных колебательных системах всегда есть потери, поэтому свободные колебания являются *затухающими*. *Резонансная кривая* — графическое представление зависимости амплитуды вынужденных колебаний системы от частоты внешнего периодического воздействия. Следовательно, по оси абсцисс откладывается частота (f), а по оси ординат, как правило, принято откладывать не просто амплитуду, а амплитуду, нормированную на максимальное значение ($U/U_{\max}$), поэтому резонансу соответствует 1.

Допустим, у нас есть исходные данные для построения резонансной кривой изгибных колебаний стального стержня. Надеемся, что вы, как уже опытные программисты, частоту внешнего воздействия меняли с постоянным шагом δf , т. е. у вас получился эквидистантный ряд частот. Внесём эти данные в программу в виде двух массивов и построим график $U(f)/U_{\max}$. Не забудем импортировать модули `numpy` и `matplotlib`:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import rcParams
rcParams['font.size'] = 18.0
df = 0.5
f = np.arange(163.5, 174, df)
U = np.array([3.2, 3.3, 3.4, 3.5, 3.8, 4, 4.4, 4.9, 5.5,
7.4, 10.0, 4.9, 2.6, 1.4, 1.3, 1.4, 1.5, 1.7, 1.8, 1.9, 2.0])
U = U/U.max()
```

```
plt.figure(figsize = (14, 10))
plt.plot(f, U, 'o', color = 'black')
plt.xlabel(r'$f$, Гц')
plt.ylabel(r'$U/U_{\max}$')
plt.grid()
plt.show()
```

В результате получим рис. 10.1, *a*. В 2000-е гг. автоматические графопостроители для такой картинки сделали бы *кусочно-линейную интерполяцию*, т. е. соединили точки попарно прямыми. Но такая интерполяция лишена свойства гладкости. Существует *интерполяция Лангранжа* — поиск многочлена n -й степени, проходящего точно через заданные точки. Но интерполяция одним многочленом высокой степени давала значительные нежелательные колебания. Чтобы решить обе эти проблемы одновременно, была разработана техника *сплайнов* — кусочных функций, где каждый интервал известных данных интерполирован собственным многочленом. Точки исходных измерений называются *узлами* сплайна. Если сплайн имеет степень выше 1, в узлах накладываются дополнительные требования существования производных, поэтому, повышая степень, можно повышать его гладкость.

Для использования сплайнов в нашей программе необходимо из модуля `scipy` использовать библиотеку `interpolate`. Понадобится функция `splrep`, которая вычисляет коэффициенты сплайна, проходящего через заданные узлы. Функция принимает два обязательных аргумента: два массива координат узлов сплайна (f, U). Есть ещё необязательный именованный аргумент k — степень сплайна, $1 \leq k \leq 5$. По умолчанию $k = 3$, т. е. используется кубический сплайн. Функция `splrep` возвращает кортеж (t, c, k) , содержащий узлы, коэффициенты и порядок сплайна. Понадобится функция `splev` из той же библиотеки. Она позволяет вычислять значения сплайна в конкретных точках. Функция `splev` имеет два обязательных аргумента: массив абсцисс (у нас это `fn`), в которых должно быть возвращено значение сплайна (в данной

программе эти точки идут в 100 раз чаще, чем исходные замеры, т. е. через 0,005 Гц), и кортеж, возвращаемый с помощью функции `splrep`, содержащий узлы, коэффициенты и степень сплайна. Функция `splev` возвращает массив значений сплайна в точках `fn`. В данной программе это массив `Un`:

```
tck = splrep(f, U)
fn = np.arange(f[0], f[-1], df*0.01)
Un = splev(fn, tck)
plt.plot(fn, Un, color = 'grey')
```

Далее осталось найти значение *резонансной частоты*, исходя из того, что этому значению соответствует максимум сплайна, и вывести её в заголовке итогового графика. У массивов `numpy` есть метод `argmax`, который возвращает индекс максимального значения в массиве. Применим этот метод к массиву `Un`, а затем найдём в массиве `fn` элемент с этим индексом — это и будет резонансная частота. Оценка погрешности аппроксимации сплайнами — в настоящее время не до конца решённая научная проблема прикладной математики. Для практических целей можно использовать приближённую оценку, основанную на степени k используемого

сплайна: $\Delta f_r = \frac{\delta f}{(k+1)!}$ (следует помнить, что эта оценка предпола-

гает, что все исходные данные измерены точно, а число узлов стремится к бесконечности). Понадобится ещё функция `factorial` из библиотеки `math` модуля `numpy`, а также написанная нами функция `roundoff`. Итоговый результат представлен на рис. 10.1, б:

```
fr = fn[Un.argmax()]
dfr = df/np.math.factorial(tck[2]+1)
plt.title('Резонансная кривая\n' + r'$f_r=$' +
str(roundoff(fr,dfr)[0]) + r'$\pm$' +
str(roundoff(fr,dfr)[1]) + ' Гц')
```

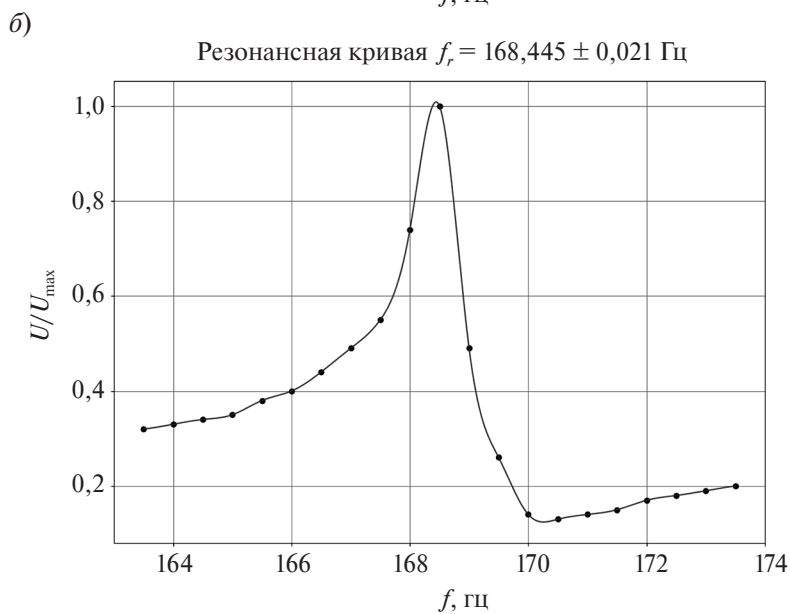
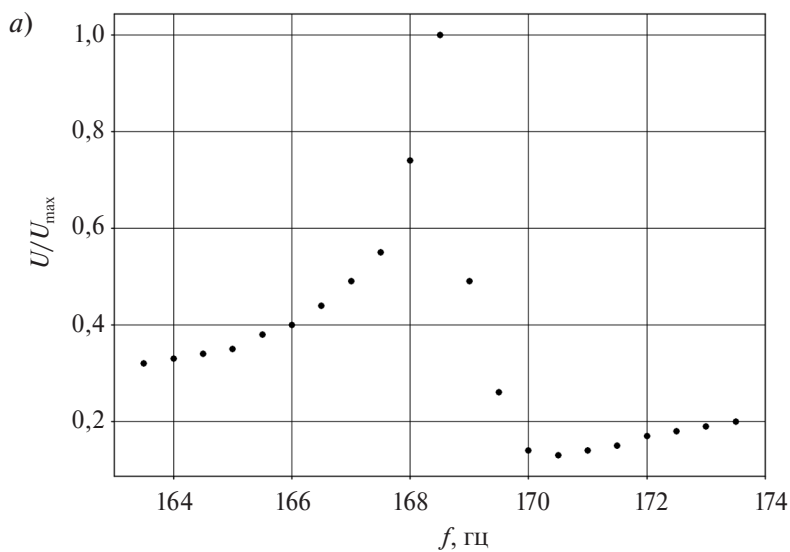


Рис. 10.1. Исходные данные для резонансной кривой (a);
интерполяция сплайнами (б)

11. ОЦЕНКА ПОГРЕШНОСТЕЙ КОСВЕННЫХ ИЗМЕРЕНИЙ

В лабораторной работе «Вынужденные изгибные колебания стального стержня», используя вычисленную резонансную частоту, необходимо по формуле (11.1) рассчитать *модуль Юнга* – физическую величину, характеризующую способность материала сопротивляться растяжению или сжатию при упругой деформации:

$$E = \frac{64 \cdot L^4 \cdot \rho \cdot f_r^2}{\pi^2 \cdot D^2}, \quad (11.1)$$

где L – длина стержня; D – диаметр стержня; ρ – плотность материала стержня; f_r – резонансная частота изгибных колебаний.

Данная задача представляет для нас интерес с точки зрения вычисления погрешности косвенного измерения – модуля Юнга. *Косвенными* называют измерения, полученные в результате расчётов, использующих результаты *прямых* измерений физических величин. В общем виде можно записать, что при косвенных измерениях искомая величина z определяется зависимостью $z = f(a, b, c...)$, где $a, b, c...$ – прямо измеряемые величины. При этом за измеренное значение принимается значение $\bar{z} = f(\bar{a}, \bar{b}, \bar{c}...)$, где $\bar{a}, \bar{b}, \bar{c}...$ – средние значения прямо измеряемых величин.

Абсолютные погрешности прямо измеренных величин либо *явно* даны на стенде, либо в описании к работе. Например, на стенде написано: «Длина стержня $L = 325 \pm 1$ мм». Следовательно, абсолютная погрешность $\Delta L = 1$ мм. Погрешность может быть указана *неявно*, например, для плотности материала стержня она составляет единицу младшего из обозначенных разрядов числового значения плотности: для $\rho = 7,85 \cdot 10^3$ кг/м³ это $\Delta \rho = 0,001 \cdot 10^3$ кг/м³. Наконец, погрешность можно взять, используя *приборную погрешность*, например, диаметр стержня D необходимо измерить самостоятельно микрометром с ценой деления 0,01 мм. Следовательно, $\Delta D = 0,005$ мм, так как микрометр – аналоговый измерительный прибор, значит, погрешность равна половине цены его наименьшего деления.

Формула для расчёта абсолютной погрешности искомой величины z как функция погрешностей прямо измеренных величин выглядит следующим образом (11.2):

$$\Delta z = \sqrt{\left(\frac{\partial f}{\partial a} \Delta a\right)^2 + \left(\frac{\partial f}{\partial b} \Delta b\right)^2 + \left(\frac{\partial f}{\partial c} \Delta c\right)^2 + \dots} \quad (11.2)$$

где $\frac{\partial f}{\partial a}$, $\frac{\partial f}{\partial b}$, $\frac{\partial f}{\partial c} \dots$ — частные производные искомой функции z по соответствующим переменным; Δa , Δb , $\Delta c \dots$ — абсолютные погрешности прямых измерений.

Попробуем проделать необходимые вычисления для формулы модуля Юнга: $\frac{\partial E}{\partial L} = \frac{64 \cdot 4L^3 \cdot \rho \cdot f_r^2}{\pi^2 \cdot D^2} = \frac{4}{L} \bar{E}$; $\frac{\partial \rho}{\partial L} = \frac{64 \cdot L^4 \cdot f_r^2}{\pi^2 \cdot D^2} = \frac{1}{\rho} \bar{E}$;

$$\frac{\partial E}{\partial f_r} = \frac{64 \cdot L^4 \cdot \rho \cdot 2f_r}{\pi^2 \cdot D^2} = \frac{2}{f_r} \bar{E}; \quad \frac{\partial E}{\partial D} = \frac{(-2) \cdot 64 \cdot L^4 \cdot \rho \cdot f_r^2}{\pi^2 \cdot D^3} = -\frac{2}{D} \bar{E}. \text{ Тогда:}$$

$$\begin{aligned} \Delta E &= \bar{E} \cdot \sqrt{\left(4 \frac{\Delta L}{L}\right)^2 + \left(\frac{\Delta \rho}{\rho}\right)^2 + \left(2 \frac{\Delta f_r}{f_r}\right)^2 + \left(-2 \frac{\Delta D}{D}\right)^2} = \\ &= \bar{E} \cdot \sqrt{(4 \cdot \delta L)^2 + (\delta \rho)^2 + (2 \cdot \delta f_r)^2 + (-2 \cdot \delta D)^2} = \bar{E} \cdot \delta E. \end{aligned}$$

Тогда формулу (см. 11.2) можно переписать в виде (11.3):

$$\Delta z = \bar{z} \cdot \delta z. \quad (11.3)$$

Внесём в программу все исходные данные и формулы для расчётов:

```
L = 325.0*10**(-3)
dL = 1.0*10**(-3)
ro = 7.85*10**3
dro = 0.01*10**3
D = 9.35*10**(-3)
dD = 0.005*10**(-3)
E = (64*L**4*ro*fr**2) / (np.pi**2*D**2)
```

```

dE = E*((4*dL/L)**2 + (dro/ro)**2 + (2*dfr/fr)**2 +
(-2*dD/D)**2)**0.5
print(str(roundoff(a,da)[0]) + '\u00B1' +
str(roundoff(a,da)[1]))

```

В итоге получим значения модуля Юнга стального стержня $E_{\text{прак}} = (184,3 \pm 2,3) \cdot 10^9$ Па. Теоретическое значение модуля Юнга для стали $E_{\text{теор}} = 200 \cdot 10^9$ Па. Как видим, теоретическое значение не попадает в доверительный интервал экспериментально полученного значения.

12. ОПРЕДЕЛЕНИЕ ДОБРОТНОСТИ ПО ГРАФИКУ РЕЗОНАНСНОЙ КРИВОЙ

Кроме определения резонансной частоты в тех же лабораторных работах часто просят определить добротность. *Добротность* — это параметр колебательной системы, определяющий ширину резонанса и характеризующий, во сколько раз запасы энергии в системе больше, чем потери энергии за время изменения фазы на 1 радиан. Добротность обратно пропорциональна скорости затухания собственных колебаний в системе. То есть чем выше добротность колебательной системы, тем меньше потери энергии за каждый период и тем медленнее затухают колебания.

Для простых хорошо изученных колебательных систем (и механических, и электрических) существуют аналитические формулы для расчёта добротности. Если, например, в лабораторной работе используется электрический колебательный контур, то добротность можно рассчитать по такой формуле:

$$Q = \frac{L}{R_{\text{circuit}}} \sqrt{\frac{1}{CL} - \frac{R_{\text{circuit}}^2}{4L^2}},$$

где L — индуктивность катушки; C — ёмкость конденсатора; R_{circuit} — сопротивление резонансного контура. Если в контуре есть только катушка и конденсатор, то R_{circuit} фактически является сопротивлением катушки, если последовательно с катушкой подключить до-

полнительный резистор, то $R_{\text{circuit}} = R_L + R$, при этом сопротивление катушки, скорее всего, окажется пренебрежимо мало по сравнению с R . Если R, L, C даны, то задачу можно считать решённой.

Но бывают случаи, когда необходимые параметры колебательной системы неизвестны. Тогда добротность можно определить через ширину резонанса. *Ширина резонансной кривой Δf* (она же *полоса пропускания колебательного контура*) — это интервал частоты, в пределах которого амплитуда вынужденных колебаний системы отличается от резонансной амплитуды не более чем в $\sqrt{2}$ раз, а энергии колебаний отличаются не более чем в два раза. Тогда добротность вычисляется по формуле (12.1):

$$Q = \frac{f_r}{\Delta f}, \quad (12.1)$$

где $\Delta f = f_2 - f_1$, а f_1 и f_2 — частоты, при которых значение амплитуды становится в $\sqrt{2}$ меньше, чем максимальное. Напишем отдельную функцию, которая на вход будет получать массивы частот и амплитуд, для которых сделана сплайн-аппроксимация, а на выходе будет выдавать резонансную частоту, ширину полосы пропускания и значение добротности:

```
def resonance(f, U):
    for i in range(len(U)-1):
        if U[i] <= U.max()/2**0.5 and U[i+1] >
U.max()/2**0.5:
            f1 = f[i]
            if U[i] >= U.max()/2**0.5 and U[i+1] <
U.max()/2**0.5:
                f2 = f[i]
            fr = f[U.argmax()]
            df = f2-f1
            Q = fr/df
    return (fr, df, Q)
```

Далее эту функцию можно вставлять в любую программу, где требуется найти характеристики резонансной кривой. Напишите

программу для определения добротности электрического колебательного контура без дополнительного резистора R и с ним. Сделайте аппроксимацию резонансных кривых сплайнами. Вычислите резонансную частоту, ширину полосы пропускания, значение добротности и их погрешности. Вынесите все рассчитанные параметры в заголовок графика (рис. 12.1):

```
df = 0.5
f = np.arange (35, 42.5, df)
U = np.array([1.27,1.58,1.98,2.57,3.45,5.31,5.78,
4.28, 3.13,2.42,2.02,1.62,1.47,1.31,1.15])
U = U/U.max()
UR = np.array([1.19,1.39,1.58,1.86,2.14,2.38,2.46,
2.30,2.02,1.82,1.62,1.47,1.35,1.19,1.11])
UR = UR/UR.max()
tckU = splrep(f, U)
tckUR = splrep(f, UR)
fn = np.arange (f[0], f[-1], df*0.01)
Un = splev(fn, tckU)
UnR = splev(fn, tckUR)
fr, Df, Q = resonance(fn,Un)
dfr = df/np.math.factorial(tckU[2]+1)
dDf = 2*dfr
dQ = Q*((dfr/fr)**2+(dDf/Df)**2)**0.5
frR, DfR, QR = resonance(fn,UnR)
dfrR = df/np.math.factorial(tckUR[2]+1)
dDfR = 2*dfrR
dQR = QR*((dfrR/frR)**2+(dDfR/DfR)**2)**0.5
plt.figure(figsize = (14, 10))
plt.plot(f, U, 'o', color='black', label='LC-контур')
plt.plot(f, UR, 's', color='black', label='RLC-контур')
plt.plot(fn, Un, fn, UnR, color = 'grey')
plt.title('Резонансные кривые\n' + 'LC-контур:
' + r'$f_r$' + str(roundoff(fr,dfr)[0]) +
r'$\pm$' + str(roundoff(fr,dfr)[1]) + ' кГц,
```

```

` + r'$\Delta f=$' + str(roundoff(Df,dDf)
[0]) + r'$\pm$' + str(roundoff(Df,dDf)[1]) + `
кГц, Q=' + str(roundoff(Q,dQ)[0]) + r'$\pm$'
+ str(roundoff(Q,dQ)[1]) + ` \nRLC-контур: ` +
r'$f_r=$' + str(roundoff(frR,dfrR)[0]) + r'$\pm$'
+ str(roundoff(frR,dfrR)[1]) + ` кГц, ` +
r'$\Delta f=$' + str(roundoff(DfR,dDfR)[0]) +
r'$\pm$' + str(roundoff(DfR,dDfR)[1]) + ` кГц,
Q=' + str(roundoff(QR,dQR)[0]) + r'$\pm$' +
str(roundoff(QR,dQR)[1]))
plt.xlabel(r'$f$, кГц')
plt.ylabel(r'$U/U_{\max}$')
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()

```

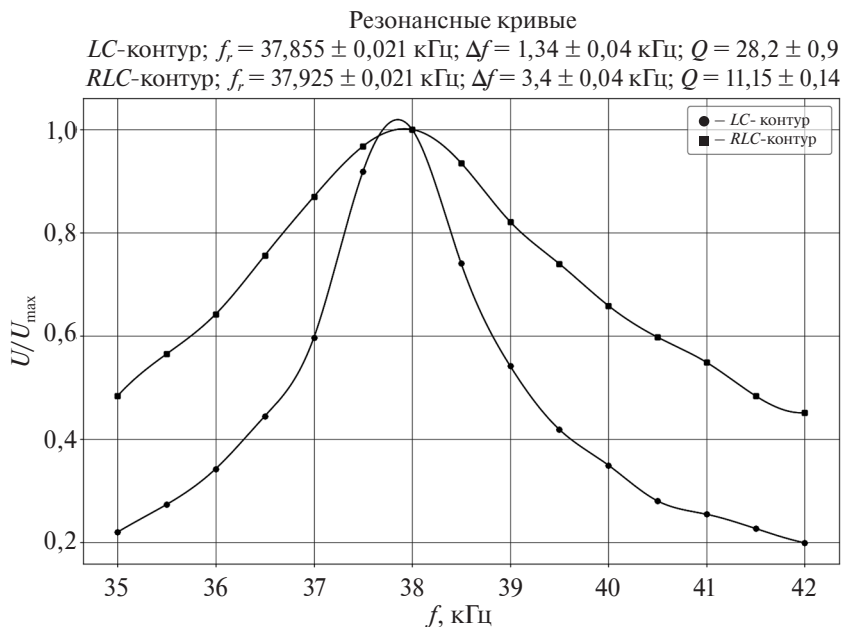


Рис. 12.1. Резонансные кривые *LC*- и *RLC*-контуров

13. ОПРЕДЕЛЕНИЕ ДОБРОТНОСТИ ПО ГРАФИКУ КРИВОЙ ЗАТУХАНИЯ

Существует ещё один способ определения добротности колебательной системы. Резонансные кривые строились для случая вынужденных колебаний. Рассмотрим *свободные затухающие колебания* — колебания, амплитуда которых со временем уменьшается из-за потери энергии колебательной системы.

В ходе выполнения лабораторной работы на экране осциллографа фиксировались положения максимумов затухающих синусоидальных колебаний (t , Y) для LC - и RLC -контуров. Запишем эти координаты в программу, затем амплитуду колебаний отнормируем на максимальное значение (как для резонансных кривых делали) и отложим получившиеся нормированные данные (t , $Y/Y_{\max}$) точками на графике (рис. 13.1):

```
dt = 26.5
time = np.arange(0, 150, dt)
y = np.array([148, 132, 118, 106, 94, 86])
y = y/y.max()
yR = np.array([126, 94, 70, 52, 40, 30])
yR = yR/yR.max()
plt.figure(figsize = (14, 10))
plt.plot(time, y, 'o', color='black', label='LC-
контур')
plt.plot(time, yR, 's', color='black', label='RLC-
контур')
plt.xlabel(r'$t$, мкс$')
plt.ylabel(r'$Y/Y_{\max}$')
plt.grid()
plt.legend()
plt.show()
```

Ряд времени получился эквидистантным, поэтому сразу можно зафиксировать, что период свободных колебаний $T = dt$ мкс.

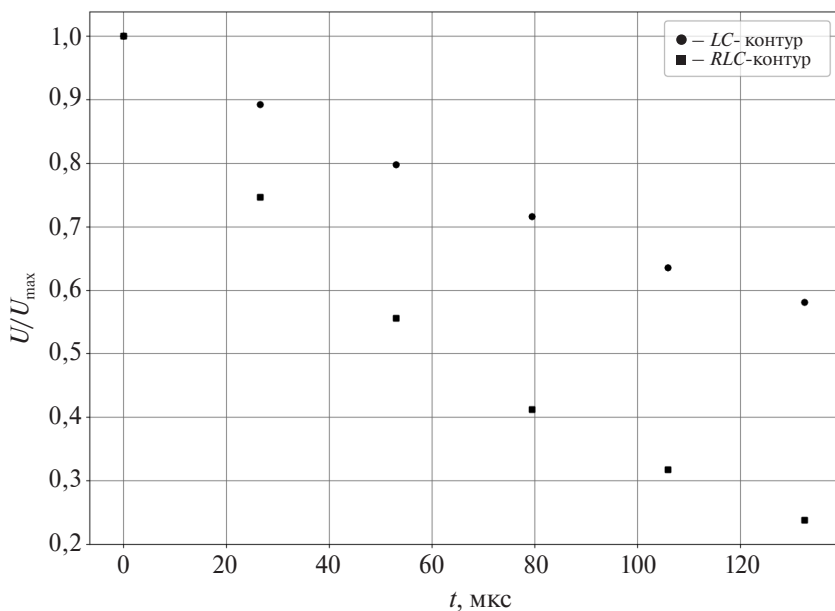


Рис. 13.1. Исходные данные для кривых затухания

При этом данные замеры проводились с помощью цифрового осциллографа, у которого цена деления по оси времени составляла $\Delta T = 0,01$ мкс (именно в данном эксперименте, в общем эта величина меняется динамически, для данного эксперимента следует отслеживать отдельно). Резонансная частота равна $f_r = \frac{1}{T}$ МГц. Погрешность оценки резонансной частоты будет рассчитываться по формуле

$$\Delta f_r = f \cdot \frac{\Delta T}{T}.$$

$$T = dt$$

$$dT = 0.01$$

$$f_r = 1/T$$

$$df_r = f_r \cdot dT/T$$

Далее необходимо по формуле (13.1) определить *логарифмический декремент затухания* — безразмерную физическую величину, описывающую уменьшение амплитуды колебательного процесса и равную натуральному логарифму отношения двух последовательных (отстоящих друг от друга на характерный период колебания) амплитуд колеблющейся величины в одну и ту же сторону:

$$\lambda = \ln \frac{Y(t)}{Y(t+T)}. \quad (13.1)$$

Для большей точности расчётов лучше всего логарифмический декремент затухания рассчитать для всех имеющихся пар максимумов амплитуд. Тогда у нас будет массив декрементов, для которого можно вычислить выборочное среднее (8.3) и доверительный интервал оценки среднего (9.2):

```
lyam = np.empty(len(y)-1)
for i in range(len(y)-1):
    lyam[i] = log(y[i]/y[i+1])
N = len(lyam)
lyam_m = np.mean(lyam)
lyam_s = np.std(lyam, ddof=1)/N**0.5
lyamR = np.empty(len(yR)-1)
for i in range(len(yR)-1):
    lyamR[i] = log(yR[i]/yR[i+1])
lyamR_m = np.mean(lyamR)
lyamR_s = np.std(lyamR, ddof=1)/N**0.5
tan = t.ppf(1-p/2, N-1)
dlyam = lyam_s*tan
dlyamR = lyamR_s*tan
```

Добротность контура через логарифмический декремент затухания рассчитывается по формуле (13.2):

$$Q = \frac{\pi}{\lambda}. \quad (13.2)$$

Для расчётов будем использовать $\langle \lambda \rangle$, а погрешность Q оценивать так, как погрешность косвенных измерений:

$$\Delta Q = Q \frac{\Delta \lambda}{\langle \lambda \rangle}.$$

```
Q = np.pi/lyam_m
dQ = Q*dlyam/lyam_m
QR = np.pi/lyamR_m
dQR = QR*dlyamR/lyamR_m
```

Свободные затухающие колебания описываются уравнением (13.3):

$$Y = Y_{\max} e^{-\beta t} \cos(\omega t), \quad (13.3)$$

где β — коэффициент затухания колебаний, который вычисляется по формуле (13.4); ω — угловая скорость (она же круговая частота) колебаний, которая вычисляется по формуле (13.5); $e^{-\beta t}$ — огибающая затухающих колебаний (она же кривая затухания):

$$\beta = \frac{\lambda}{T}; \quad (13.4)$$

$$\omega = \frac{2\pi}{T}. \quad (13.5)$$

Погрешности для β и ω считаются как погрешности косвенных измерений $\Delta\beta = \beta \cdot \sqrt{\left(\frac{\Delta\lambda}{\lambda}\right)^2 + \left(\frac{\Delta T}{T}\right)^2}$ и $\Delta\omega = \omega \cdot \frac{\Delta T}{T}$, а погрешность периода колебаний берется из приборной погрешности $\Delta T = 0,01$ мкс.

Реконструируем (реконструкция математической модели — это процедура получения модельных уравнений по экспериментальным временным рядам) затухающие колебания и кривые затухания, используя вычисленные значения T , β и ω . Построим получившиеся зависимости на графике. Для наглядности можно построить эти зависимости в области, где замеры не производились, т. е. сделать экстраполяцию. Для этого можно

последнее значение нового массива времени сделать, например, в три раза больше, чем исходного массива. Традиционно выведем в заголовок графика все найденные в программе величины (рис. 13.2):

```

beta = lyam_m/T
dbeta = beta*((dlyam/lyam_m)**2+(dT/T)**2)**0.5
betaR = lyamR_m/T
dbetaR = betaR*((dlyamR/lyamR_m)**2+(dT/T)**2)**0.5
w = 2*np.pi/T
dw = w*dt/T
tn = np.arange (time[0], time[-1]*3, dt*0.01)
e = np.exp(-beta*tn)
eR = np.exp(-betaR*tn)
s = e*np.cos(w*tn)
sR = eR*np.cos(w*tn)
plt.plot(tn, e, '--', tn, eR, '--', color = 'grey')
plt.plot(tn, s, tn, sR, color = 'grey')
p l t . t i t l e ( r ' $ f _ r = $ ' +
str(roundoff(fr*10**3,dfr*10**3)[0]) + r'$\pm$'
+ str(roundoff(fr*10**3,dfr*10**3)[1]) + 'кГц' +
r'$;~\omega=$' + str(roundoff(w,dw)[0]) + r'$\pm$'
+ str(roundoff(w,dw)[1]) + ' рад/с\nLC-контур:
' + r'$\lambda=$' + str(roundoff(lyam_m,dlyam)
[0]) + r'$\pm$' + str(roundoff(lyam_m,dlyam)
[1]) + r'$;~\beta=$' + str(roundoff(beta,dbeta)
[0]) + r'$\pm$'+ str(roundoff(beta,dbeta)[1]) +
'; Q = ' + str(roundoff(Q,dQ)[0]) + r'$\pm$' +
str(roundoff(Q,dQ)[1]) + '\nRLC-контур: ' + r'$\lambda=$'
+ str(roundoff(lyamR_m,dlyamR)[0]) +
r'$\pm$' + str(roundoff(lyamR_m,dlyamR)[1]) +
r'$;~\beta=$' + str(roundoff(betaR,dbetaR)[0])
+ r'$\pm$' + str(roundoff(betaR,dbetaR)[1]) + ';
Q = ' + str(roundoff(QR,dQR)[0]) + r'$\pm$' +
str(roundoff(QR,dQR)[1]))

```

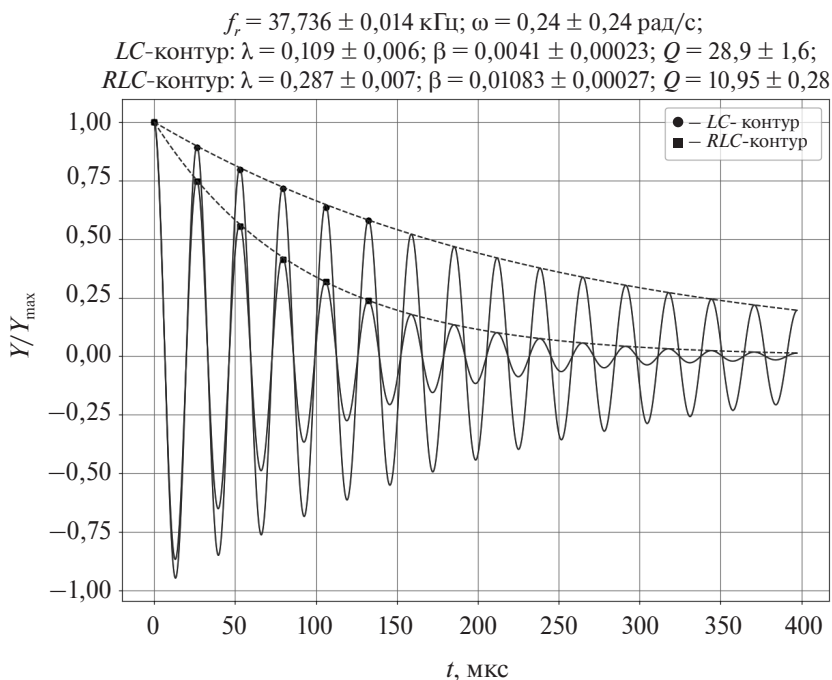


Рис. 13.2. Реконструкция затухающих колебаний и кривых затухания

В конце сравните величины резонансной частоты и добротности, полученные двумя способами: с помощью резонансной кривой и кривой затухания (табл. 13.1).

Таблица 13.1

Сравнение двух методов определения характеристик резонанса

№ п/п	Резонансная кривая (вынужденные колебания)		Кривая затухания (свободные колебания)	
	$f_r, \text{ кГц}$	Q	$f_r, \text{ кГц}$	Q_λ
$LC\text{-контур}$	$37,855 \pm 0,021$	$28,2 \pm 0,9$	$37,736 \pm 0,014$	$28,9 \pm 1,6$
$RLC\text{-контур}$	$37,925 \pm 0,021$	$11,15 \pm 0,14$	$37,736 \pm 0,014$	$10,95 \pm 0,28$

С математической точки зрения можно записать, что и для LC -, и для RLC -контура множества Q и Q_λ пересекаются: $Q \cap Q_\lambda$.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

МИ 2083–90. ГСИ. Измерения косвенные. Определение результатов измерений и их погрешностей. — 1990. — 10 с.

Сысоева М. В. Программирование для «нормальных» с нуля на языке Python : учебник. В 2 ч. Ч. 1 / М. В. Сысоева, И. В. Сысоев ; отв. ред. В. Л. Черный. — 2-е изд., испр. и доп. — М. : ООО «МАКС Пресс», 2023. — 184 с. — (Библиотека ALT).

Сысоева М. В. Программирование для «нормальных» с нуля на языке Python : учебник. В 2 ч. Ч. 2 / М. В. Сысоева, И. В. Сысоев ; отв. ред. В. Л. Черный. — М. : ООО «МАКС Пресс», 2023. — 184 с. — (Библиотека ALT).

Фридман А. Э. Основы метрологии : современный курс / А. Э. Фридман. — СПб. : Проффессионал, 2008. — 284 с.

ПРИЛОЖЕНИЯ

Приложение 1

**Свидетельство об официальной регистрации программы
для ЭВМ «Программа анализа и статистической обработки
результатов лабораторного эксперимента «Температурная
зависимость электрического сопротивления металла
и полупроводника» № 2026680431**

```
def coeff(x, y):
    k = (sum(x*y) - sum(x) * sum(y) / len(y)) / (sum(x**2) -
sum(x)**2 / len(y))
    b = 1 / len(y) * (sum(y) - k * sum(x))
    return(k, b)
def approx_error(x, y):
    k, b = coeff(x, y)
    e = y - (k*x + b)
    return np.var(e, ddof=2)
def coeff_error(x, y):
    ke = (approx_error(x, y) / sum((x - np.
mean(x))**2))**0.5
    be = ((approx_error(x, y) * sum(x**2)) /
(len(x) * sum((x - np.mean(x))**2)))**0.5
    return(ke, be)
def roundoff(a_original, da_original):
    da = da_original
    n = 0
    if abs(da) >= 10:
        while abs(da) >= 10:
            da = da / 10
            if da >= 3:
                n = n - 1
    elif abs(da) < 1:
        while abs(da) < 1:
```

```

        da = da*10
        n = n + 1
        if (da>=3) and (da<10):
            break
        elif (da>=1) and (da<3):
            n = n + 1
    else:
        if int(da)>=3:
            n = 0
        else:
            n = n + 1
        return (round(a_original,n), round(da_
original,n))

from math import *
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import rcParams
rcParams['font.size'] = 18.0
kb = 1.38*10**(-23) #постоянная Больцмана
tempr = np.arange(25,73,3,dtype=float)
T = tempr + 273.15
Rm = np.array([99.1,99.5,100.0,101.0,102.3,103.5,104.8, 106.
1,107.0,108.8,110.7,112.4,114.2,114.9,115.7,116.6
])
k, b = coeff(tempr,Rm)
x = np.arange(-1,73,0.5)
y = k*x + b
a = k/b
dk, db = coeff_error(tempr,Rm)
da = a*((dk/k)**2+(db/b)**2)**0.5
plt.figure(figsize = (14, 10))
plt.subplot(2,1,1)
plt.plot(tempr, Rm, 'o', color = 'black', label =

```

```

`Исходные данные')
plt.plot(x, y, color = 'grey', label = 'Аппроксимирующая прямая')
plt.title('Зависимость сопротивления металла от температуры\n' + 'y = ' + str(round(k,3)) + 'x+' + str(round(b,3)) + '\nТемпературный коэффициент сопротивления металла ' + r'$\alpha$ = ' + str(roundoff(a,da)[0]) + r'$\pm$' + str(roundoff(a,da)[1]) + r'$\sim K^{-1}$')
plt.legend()
plt.xlabel(r'$t$, ^o$C$')
plt.ylabel(r'$R_m$, кОм')
plt.grid()
Rs = np.array([48.3,45.1,42.5,38.7,34.4,30.4,26.6,23.7,21.2,18.8,16.4,14.4,13.1,12.3,11.5,10.6])
Rs = Rs*10**3
k, b = coeff(1000/T,np.log(Rs))
x = 10**3/T
y = k*x + b
Eg = 2*kb*k*10**3
dk, db = coeff_error(1000/T,np.log(Rs))
dEg = 2*kb*dk*10**3
plt.subplot(2,1,2)
plt.plot(1000/T, np.log(Rs), 'o', color = 'black', label = 'Исходные данные')
plt.plot(x, y, color = 'grey', label = 'Аппроксимирующая прямая')
plt.title('Зависимость сопротивления полупроводника от температуры\n' + 'y = ' + str(round(k*10**3,3)) + 'x' + str(round(b,3))+' \nШирина запрещённой зоны полупроводника\n' + r'$E_g$ = ' + str(roundoff(Eg*10**18, dEg*10**18)[0]) + r'$\cdot 10^{-18}$\pm$' + str(roundoff(Eg*10**18, dEg*10**18)[1]) + ' Дж = ' + str(roundoff(Eg*6

```

```

.242*10**18,dEg*6.242*10**18)[0]) + r'$\pm$' +
str(roundoff(Eg*6.242*10**18, dEg*6.242*10**18)
[1]) + ' эВ')
plt.xlabel(r'$1000/T, K^{-1}$')
plt.ylabel(r'$\ln R_s$')
plt.grid()
plt.tight_layout()
plt.show()

```

Приложение 2

Свидетельство об официальной регистрации программы для ЭВМ «Программа анализа и статистической обработки результатов лабораторного эксперимента «Моделирование случайной величины и исследование её распределения» № 2026681302

```

def roundoff(a_original, da_original):
    da = da_original
    n = 0
    if abs(da)>=10:
        while abs(da)>=10:
            da = da/10
            if da>=3:
                n = n - 1
    elif abs(da)<1:
        while abs(da)<1:
            da = da*10
            n = n + 1
            if (da>=3) and (da<10):
                break
            elif (da>=1) and (da<3):
                n = n + 1
    else:
        if int(da)>=3:
            n = 0
        else:

```

```

        n = n + 1
    return (round(a_original,n), round(da_
original,n))
import matplotlib.pyplot as plt
import numpy as np
from decimal import Decimal, ROUND_UP
from matplotlib import rcParams
rcParams['font.sans-serif']=['Liberation Sans',
'Arial']
rcParams['font.size'] = 18.0
from scipy.stats import t
a = 0.68; p = 1 - a
time = np.array([5.07,5.30,5.06,4.92,5.38,4.74,
4.87,4.92, 5.01,4.86,4.57,5.07,5.00,5.52,5.29,4.88
,5.06,4.74,5.07, 5.38,4.80,4.95,4.73,5.21,5.24,4.6
2,5.11,4.63,4.97,4.40, 4.87,4.93,4.88,4.99,4.90,5.
07,5.36,5.10,4.47,5.30,4.74, 4.69,5.35,5.32,4.81,4
.88,4.45,4.76,5.07,5.32])
N = len(time)
plt.figure(figsize = (14, 10))
nbins = int(N**0.5)
delta_t = float(Decimal((np.max(time) -
np.min(time))/nbins).quantize(Decimal(str(1e-2)),
rounding=ROUND_UP))
medt = (np.max(time) + np.min(time))/2
poldiap = (np.max(time) - np.min(time))/2
plt.hist(time, bins = np.linspace(medt-
poldiap, medt+poldiap, nbins+1), color='white',
edgecolor='black', density=True)
tm = np.mean(time)
ts = np.std(time, ddof=1)
tst = ts/(N**0.5)
dt = 0.01
tt = np.arange(np.min(time), np.max(time)+dt, dt)

```

```

rho=(np.exp(-1*(tt-tm)**2/(2*ts**2)))/(2*np.
pi)**0.5*ts)
tan = t.ppf(1-p/2, N-1)
plt.plot(tt, rho, color = 'grey')
plt.xlabel('t, c')
plt.ylabel(r'$\Delta N/N\Delta t$')
plt.title(r'$\Delta t = $' + str(delta_t) + r'$, \sigma_t = $' + str(round(ts,3)) + r'$, \sigma_{\angle t \angle} = $' + str(round(tst,3)) + r'$, \rho_{max} = $' + str(round(0.4/ts,3)) + r'$, \overline{t} = ($' + str(roundoff(tm,tst*tan)[0]) + r'$\pm$' + str(roundoff(tm,tst*tan)[1]) + ') при '+r'$\alpha = $' + str(a))
plt.show()

```

Приложение 3

**Свидетельство об официальной регистрации программы
для ЭВМ «Программа анализа и статистической обработки
результатов лабораторного эксперимента «Вынужденные изгибные
колебания стального стержня» № 2026681305**

```

def roundoff(a_original, da_original):
    da = da_original
    n = 0
    if abs(da)>=10:
        while abs(da)>=10:
            da = da/10
            if da>=3:
                n = n - 1
    elif abs(da)<1:
        while abs(da)<1:
            da = da*10
            n = n + 1
            if (da>=3) and (da<10):
                break

```

```

        elif (da>=1) and (da<3):
            n = n + 1
    else:
        if int(da)>=3:
            n = 0
        else:
            n = n + 1
        return (round(a_original,n), round(da_
original,n))

```

```

import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import splrep, splev, PPoly
from scipy.stats import t
from matplotlib import rcParams
rcParams['font.size'] = 18.0

```

```

L = 325.0*10**(-3)
dL = 1.0*10**(-3)
ro = 7.85*10**3
dro = 0.01*10**3
D = 9.35*10**(-3)
dD = 0.005*10**(-3)
df = 0.5
f = np.arange(163.5, 174, df)
U = np.array([3.2,3.3,3.4,3.5,3.8,4,4.4,4.9,
5.5,7.4,10.0, 4.9,2.6,1.4,1.3,1.4,1.5,1.7,1.8,1.9,
2.0])
U = U/U.max()
tck = splrep(f, U)
fn = np.arange (f[0], f[-1], df*0.01)
Un = splev(fn, tck)
fr = fn[Un.argmax()]
dfr = df/np.math.factorial(tck[2]+1)
E = (64*L**4*ro*fr**2)/(np.pi**2*D**2)

```

```

dE = E*((4*dL/L)**2 + (dro/ro)**2 + (2*dfr/fr)**2
+      (-2*dD/D)**2)**0.5
plt.figure(figsize = (14, 10))
plt.plot(f, U, 'o', color = 'black')
plt.plot(fn, Un, color = 'grey')
plt.title('Резонансная кривая\n' + r'$f_{r}$'
r='$' + str(roundoff(fr,dfr)[0]) + r'$\pm$'
+ str(roundoff(fr,dfr)[1]) + ' Гц; E=' +
str(roundoff(E/10**9,dE/10**9)[0]) + r'$\pm$' +
str(roundoff(E/10**9,dE/10**9)[1]) + r'$\cdot$
10^9$ Па')
plt.xlabel(r'$f$, Гц')
plt.ylabel(r'$U/U_{max}$')
plt.grid()
plt.show()

```

Приложение 4

Свидетельство об официальной регистрации программы для ЭВМ «Программа анализа и статистической обработки результатов лабораторного эксперимента «Исследования электромагнитных колебаний в параллельных LC- и RLC-контурах» № 2026681330

```

def resonance(f,U):
    for i in range(len(U)-1):
        if U[i] <= U.max()/2**0.5 and U[i+1] >
U.max()/2**0.5:
            f1 = f[i]
            if U[i] >= U.max()/2**0.5 and U[i+1] <
U.max()/2**0.5:
                f2 = f[i]
    fr = f[U.argmax()]
    Df = f2-f1
    Q = fr/Df
    return (fr, Df, Q)

```

```

def roundoff(a_original, da_original):
    da = da_original
    n = 0
    if abs(da)>=10:
        while abs(da)>=10:
            da = da/10
            if da>=3:
                n = n - 1
    elif abs(da)<1:
        while abs(da)<1:
            da = da*10
            n = n + 1
            if (da>=3) and (da<10):
                break
            elif (da>=1) and (da<3):
                n = n + 1
    else:
        if int(da)>=3:
            n = 0
        else:
            n = n + 1
    return (round(a_original,n), round(da_
original,n))

```

```

from math import *
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import splev, splrep
from scipy.stats import t
from matplotlib import rcParams
rcParams['font.size'] = 18.0

```

```

a = 0.68 #доверительная вероятность
p = 1 - a #уровень значимости

```

```

dt = 26.5
T = dt
dT = 0.01
fr = 1/T
dfr = fr*dT/T
time = np.arange(0, 150, dt)
y = np.array([148,132,118,106,94,86])
y = y/y.max()
yR = np.array([126,94,70,52,40, 30])
yR = yR/yR.max()
df = 0.5
f = np.arange (35, 42.5, df)
U      =      np.array([1.27,1.58,1.98,2.57,3.45,5.
31,5.78,4.28, 3.13,2.42,2.02,1.62,1.47,1.31,1.15])
U = U/U.max()
UR     =     np.array([1.19,1.39,1.58,1.86,2.14,2.38,
2.46,2.30, 2.02,1.82,1.62,1.47,1.35,1.19,1.11])
UR = UR/UR.max()

#Резонансная кривая
tckU = splrep(f, U)
tckUR = splrep(f, UR)
fn = np.arange (f[0], f[-1], df*0.01)
Un = splev(fn, tckU)
UnR = splev(fn, tckUR)
fr, Df, Q = resonance(fn,Un)
dfr = df/np.math.factorial(tckU[2]+1)
dDf = 2*dfr
dQ = Q*((dfr/fr)**2+(dDf/Df)**2)**0.5
frR, DfR, QR = resonance(fn,UnR)
dfrR = df/np.math.factorial(tckUR[2]+1)
dDfR = 2*dfrR
dQR = QR*((dfrR/frR)**2+(dDfR/DfR)**2)**0.5
plt.figure(figsize = (14, 10))

```

```

plt.plot(f, U, 'o', color = 'black', label = 'LC-
контур')
plt.plot(f, UR, 's', color = 'black', label = 'RLC-
контур')
plt.plot(fn, Un, fn, UnR, color = 'grey')
plt.title('Резонансные кривые\n'+
        'LC-контур: '+ r'$f_r = $' +
        str(roundoff(fr,dfr)[0])+r'$\pm$'+
        str(roundoff(fr,dfr)[1]) +' кГц, '+r'$\
Delta f = $'+
        str(roundoff(Df,dDf)[0])+r'$\pm$'+
        str(roundoff(Df,dDf)[1])+ ' кГц, Q = '+
        str(roundoff(Q,dQ)[0])+r'$\pm$'+
        str(roundoff(Q,dQ)[1])+
        '\nRLC-контур: '+ r'$f_r = $' +
        str(roundoff(frR,dfrR)[0])+r'$\pm$'+
        str(roundoff(frR,dfrR)[1]) +' кГц, '+r'$\
Delta f = $'+
        str(roundoff(DfR,dDfR)[0])+r'$\pm$'+
        str(roundoff(DfR,dDfR)[1])+ ' кГц, Q = '+
        str(roundoff(QR,dQR)[0])+r'$\pm$'+
        str(roundoff(QR,dQR)[1]))
plt.xlabel(r'$f$, кГц$')
plt.ylabel(r'$U/U_{max}$')
plt.grid()
plt.legend()
plt.tight_layout()
plt.show()

#Кривая затухания
lyam = np.empty(len(y)-1)
for i in range(len(y)-1):
    lyam[i] = log(y[i]/y[i+1])
N = len(lyam)

```

```

lyam_m = np.mean(lyam)
lyam_s = np.std(lyam, ddof=1)/N**0.5
lyamR = np.empty(len(yR)-1)
for i in range(len(yR)-1):
    lyamR[i] = log(yR[i]/yR[i+1])
lyamR_m = np.mean(lyamR)
lyamR_s = np.std(lyamR, ddof=1)/N**0.5
tan = t.ppf(1-p/2, N-1)
dlyam = lyam_s*tan
dlyamR = lyamR_s*tan
Q = np.pi/lyam_m
dQ = Q*dlyam/lyam_m
QR = np.pi/lyamR_m
dQR = QR*dlyamR/lyamR_m
beta = lyam_m/T
dbeta = beta*((dlyam/lyam_m)**2+(dT/T)**2)**0.5
betaR = lyamR_m/T
dbetaR = betaR*((dlyamR/lyamR_m)**2+(dT/T)**2)**0.5
w = 2*np.pi/T
dw = w*dt/T
tn = np.arange (time[0], time[-1]*3, dt*0.01)
e = np.exp(-beta*tn)
eR = np.exp(-betaR*tn)
s = e*np.cos(w*tn)
sR = eR*np.cos(w*tn)
plt.figure(figsize = (14, 10))
plt.plot(time, y, 'o', color = 'black', label =
'LC-контур')
plt.plot(time, yR, 's', color = 'black', label =
'RLC-контур')
plt.plot(tn, e, '--', tn, eR, '--', color = 'grey')
plt.plot(tn, s, tn, sR, color = 'grey')
plt.title(r'$f_r$'+
str(roundoff(fr*10**3,dfr*10**3)[0])+r'$\
pm$'+

```

```

        str(roundoff(fr*10**3,dfr*10**3)[1])+
κΓΠ'+r'$;~\omega=$'+
        str(roundoff(w,dw)[0])+r'$\pm$'+
        str(roundoff(w,dw)[1])+ ' пад/с\nLC-контур:
'+r'$\lambda=$'+
        str(roundoff(lyam_m,dlyam)[0])+r'$\pm$'+
        str(roundoff(lyam_m,dlyam)[1])+r'$;~\
beta=$'+
        str(roundoff(beta,dbeta)[0])+r'$\pm$'+
        str(roundoff(beta,dbeta)[1])+'; Q = '+
        str(roundoff(Q,dQ)[0])+r'$\pm$'+
        str(roundoff(Q,dQ)[1])+'\nRLC-контур:
'+r'$\lambda=$'+
        str(roundoff(lyamR_m,dlyamR)[0])+r'$\
pm$'+
        str(roundoff(lyamR_m,dlyamR)[1])+r'$;~\
beta=$'+
        str(roundoff(betaR,dbetaR)[0])+r'$\pm$'+
        str(roundoff(betaR,dbetaR)[1])+'; Q = '+
        str(roundoff(QR,dQR)[0])+r'$\pm$'+
        str(roundoff(QR,dQR)[1]))
plt.xlabel(r'$t, мкс$')
plt.ylabel(r'$Y/Y_{\max}$')
plt.grid()
plt.legend()
plt.show()

```

Сысоева Марина Вячеславовна
Сысоев Илья Вячеславович

АНАЛИЗ И СТАТИСТИЧЕСКАЯ ОБРАБОТКА ДАННЫХ ФИЗИЧЕСКИХ ЭКСПЕРИМЕНТОВ В УСЛОВИЯХ ПОМЕХ

Учебное пособие

Редактор *Л. В. Ларионова*
Корректор *Н. Б. Цветкова*
Компьютерная верстка *Е. Г. Орловского*
Дизайн обложки *Е. И. Бурковой*

Санитарно-эпидемиологическое заключение
№ 78.01.07.953.П.001342.01.07 от 24.01.2007 г.

Подписано в печать 24.08.2026. Формат 60×84/16. Печать цифровая.
Усл. печ. л. 4,75. Тираж 50. Заказ 4563.

Отпечатано в Издательско-полиграфическом центре
Политехнического университета.
195251, Санкт-Петербург, Политехническая ул., 29.
Тел.: (812) 552-77-17; 550-40-14.