РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Компьютерная геометрия и геометрическое моделирование

Лабораторные работы

Учебное пособие

Москва

Российский университета дружбы народов

2018

УДК 004.92:004.94 (075.8)
ББК 32.973.26-018.2+22.151
К 63

Утверждено
РИС Учёного совета
Российского университета
дружбы народов

Рецензенты:
доктор технических наук, профессор, начальник отдела УИТО и СТС РУДН
К. Е. Самуйлов;
доцент, кандидат физико-математических наук, с.н.с. ЛИТ ОИЯИ *О. И. Стрельцова*

Авторский коллектив:
М. Н. Геворкян, Д. С. Кулябов, А. В. Демидова, А. В. Королькова.
К63 Компьютерная геометрия и геометрическое моделирование:
лабораторные работы : учебное пособие / М. Н. Геворкян, Д. С. Кулябов, А. В. Демидова, А. В. Королькова. — Москва : РУДН, 2018. — 119 с. : ил.

Пособие рекомендуется для проведения лабораторных работ по курсу «Компьютерная геометрия и геометрическое моделирование» для направления 02.03.01 «Математика и компьютерные науки».

УДК 004.92:004.94 (075.8)
ББК 32.973.26-018.2+22.151

ISBN 978-5-209-08879-0

Оглавление

Лабораторная работа № 1. Установка и настройка программного окружения

1.1. Цель лабораторной работы

Целью данной лабораторной работы является знакомство с основными инструментами, позволяющими разрабатывать и выполнять учебные программы на языке Python с использованием библиотек NumPy и Matplotlib.

1.2. Предварительные сведения

В данном разделе кратко излагаются сведения о языке программирования `Python` третьей версии. Почти незатронутой останется обширная стандартная библиотека, функционал языка, касающийся объектно-ориентированного подхода к программированию и т.д. Упор будет сделан только на те сведения, которые, на наш взгляд, пригодятся для математических вычислений и работы с научными библиотеками `NumPy`, `SciPy`, особенно с `Matplotlib`. Читателям, желающим изучить язык более подробно, можно порекомендовать книги [4—6; 20], а также официальную документацию [18], которая обладает обширным обучающим разделом.

1.2.1. Характеристика языка

Язык программирования `Python` [1; 18] является интерпретируемым языком общего назначения. `Python` обладает строгой динамической типизацией. Динамичность означает, что тип переменной определяется в момент её инициализации. В свою очередь строгость типизации говорит о том, что автоматическое приведение типов за редким исключением не используется.

Язык отличается крайне компактным синтаксисом и малым количеством базовых инструкций. Широкая функциональность обеспечивается обширной стандартной библиотекой и не включена в ядро языка.

Язык `Python` обрёл популярность в следующих основных областях:

- как скриптовый язык для администрирования операционных систем и создания небольших системных и прикладных консольных утилит;
- в области сетевых сервисов и приложений;
- в области научных вычислений благодаря многочисленным библиотекам, таким как `NumPy` [16], SciPy [3], SymPy [19], `Matplotlib` [14], Numba [15].

Перечисленные области не исчерпывают всех сфер применения языка, однако являются для него основными. Нас больше всего будет интересовать применения языка Python в области научных вычислений. В настоящее время `Python` составляет конкуренцию в этой сфере таким известным коммерческим математическим пакетам как `Matlab` [13], `Mathematica` [12] и `Maple` [2; 11].

Заметим также, что в настоящее время существуют две версии языка `Python` — версия 2 и версия 3. Вторая версия не развивается, и её поддержка вскоре должна полностью прекратиться, однако она все ещё пользуется популярностью. В данном пособии мы используем исключительно третью версию.

Следует подчеркнуть, что язык `Python` не стандартизирует интерпретатор. Стандартным интерпретатором является `CPython` [8], реализованный на языке `C` (C89 и частично C99). `CPython` преобразует код `Python` в байт-код, который потом выполняется на виртуальной машине. Все примеры данного пособия предполагают использование именно CPython.

Упомянем также несколько альтернативных интерпретаторов:
- Jython [10] — интерпретатор, реализованный на языке Java и `Python`(транслирует `Python`-код в Java-байткод, что даёт возможность исполнять созданные программы на виртуальных Java машинах (JVM));
- PyPy [17] — интерпретатор, реализованный на `Python` (подмножество RPython — Restricted Python), в нём поддерживается синтаксис второй версии, отличительной особенностью является улучшенная поддержка многопоточности;
- IronPython [9] — интерпретатор на `C#` (транслирует `Python`-код в байт код для платформы .NET компании Microsoft).

1.3. Задание

1. Установите дистрибутив Miniconda, библиотеки `Matplotlib` и `NumPy`, интерактивную оболочку Jupyter Notebook (см. раздел 1.4.1).
2. Напишите простейший сценарий `helloworld.py` на языке Python (см. раздел 1.4.2).
3. Познакомьтесь с интерфейсом интерактивной оболочки Jupyter (см. раздел 1.4.3).

1.4. Последовательность выполнения работы

1.4.1. Установка интерпретатора и основных библиотек

Стандартный интерпретатор `Python`, набор библиотек и базовые средства разработки можно скачать и установить с официального сайта `python.org`. Однако удобнее воспользоваться готовыми дистрибутивами, которые кроме стандартной библиотеки включают в себя набор популярных сторонних библиотек, утилиты для установки, удаления и обновления дополнительных пакетов, а также более мощные средства для разработки. Наиболее активно развивающимся дистрибутивом языка `Python` является `Anaconda Python` [7].

1. Пройдите по ссылке `https://www.anaconda.com/download/` и скачайте дистрибутив `Miniconda` для используемой вами операционной системы. Данный дистрибутив содержит минимальный набор пакетов, поэтому в процессе выполнения данной лабораторной работы вам будет необходимо установить все необходимые библиотеки.
2. Для установки Miniconda следует скачать дистрибутив `Miniconda3-latest-Linux-x86_64.sh` и запустить установку, выполнив в консоли команду
   ```
   bash Miniconda3-latest-Linux-x86_64.sh
   ```
3. В начале установки необходимо согласиться с лицензией, набрав в консоли `yes`, а в конце установки согласиться с добавлением пути до исполняемых файлов в переменную окружения `PATH`, для чего также следует ввести `yes`.
4. После завершения установки откройте новую консоль и проверьте содержимое переменной окружения `PATH` с помощью команды
   ```
   echo $PATH
   ```
   Добавился ли путь к директории `bin`?
5. Проверьте, стала ли доступна команда `conda`, которая позволяет обновлять установленные пакеты и устанавливать новые. Вам понадобится установить две сторонние библиотеки: `Matplotlib`, `NumPy`. Также рекомендуется установить интерактивную оболочку Jupyter Notebook. Для установки любого пакета следует выполнить команду
   ```
   conda install pakage_name
   ```

Здесь `pakage_name` — название пакета в нижнем регистре. Воспользуйтесь данной командой для установки `numpy`, `matplotlib` и `jupyter`.

1.4.2. Создание и запуск сценариев на языке Python

На языке Python можно создавать и выполнять файлы сценариев (скриптов) точно таким же способом, как это делается для подавляющего большинства интерпретируемых языков. Для этого необходимо создать файл с исходным кодом и передать его интерпретатору. Инструкции, содержащиеся в файле, начнут немедленно выполняться.

1. Запустите любой знакомый вам редактор исходного кода (например, nano, gedit, geany и т.д. — практически все современные редакторы поддерживают подсветку синтаксиса `Python`).
2. Создайте простейшую программу «Hello World!». Для этого наберите в редакторе
```
print('Hello World!')
```
3. Файлы необходимо сохранять в кодировке unicode-8 и с расширением `.py`. Для выполнения кода следует запустить файл командой
```
python helloworld.py
```
4. Каков результат выполнения программы? В случае возникновения ошибки убедитесь, что используете именно 3-ю версию языка Python.
5. Проверьте теперь, что необходимые для работы библиотеки были успешно установлены. Для этого создайте ещё один файл, в котором импортируйте NumPy и Matplotlib
```
import numpy as np
import matplotlib.pyplot as plt
print(np.array([1, 2, 3, 4]))
```
6. Запустите данный файл. Каков результат выполнения программы?

Для того чтобы файл с исходным кодом можно было запускать наподобие обычного исполняемого файла, не вызывая каждый раз интерпретатор явным образом, можно в самом начале файла с исходным кодом добавить комбинацию символов `#!`, которая в программировании получила название *шебанг* (shebang).

1. Откройте созданный вами на предыдущем шаге файл `helloworld.py` и добавьте в его начало следующую строку:
```
#! /usr/bin/env python3
```
2. Наделите вашего пользователя правами на выполнение файла `helloworld.py` и запустите его как исполняемый файл. Какие команды для этого следует ввести?
3. Запустите данный файл. Каков результат выполнения программы?
4. Найдите дополнительную информацию о комбинации символов `#!`. Можно ли указать иной путь до интерпретатора `Python`, чем тот, который указан в нашем примере.

1.4.3. Использование языка Python в интерактивном режиме

Способ организации кода в виде отдельных файлов удобен для создания библиотек и сценариев с чётким алгоритмом. Если же предполагается исследовательская деятельность или изучение средств языка и библиотек, то более удобным является интерактивный подход.

При интерактивном подходе пользователь вводит инструкции языка в интерактивную оболочку и может выполнять их отдельными блоками. При этом он сразу видит результат выполнения, а также ему доступны интерактивные подсказки.

1. Запустите интерактивную оболочку Jupyter notebook, для чего наберите в консоли команду

```
jupyter notebook
```

2. После выполнения команды должен открыться браузер с интерфейсом Jupyter блокнота, а в консоль будут выводиться отладочные сообщения.
3. В браузере вы должны увидеть список файлов. Содержимое какой директории отображает Jupyter при запуске?
4. Познакомьтесь с интерфейсом оболочки Jupyter, для чего создайте новый интерактивный блокнот, нажав на кнопку `New` и выбрав `Python 3`.
5. Открывшийся интерфейс позволяет создавать отдельные блоки кода и выполнять их интерактивно в произвольной последовательности. Также можно добавлять блоки, содержащие форматированные с помощью языка разметки `Markdown` комментарии, которые могут включать в себя формулы в формате LaTeX. При использовании библиотеки `Matplotlib` можно настроить отображения создаваемых изображений непосредственно внутри интерактивного блокнота.
6. Введите несколько инструкций языка `Python` в пустую ячейку и выполните их. Для запуска кода в определённой ячейке можно нажать кнопку `Run` или же использовать комбинацию клавиш `Shift + Enter`.
7. Используя справку из меню `Help`, определите комбинации клавиш для следующих действий:
   - создание новой ячейки;
   - перемещение выбранной ячейки вверх или вниз;
   - удаление выбранной ячейки.

1.5. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:

- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку цели работы;
- описание результатов выполнения задания:
  - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
  - листинги (исходный код) программ (если они есть);
  - результаты выполнения программ (текст или снимок экрана в зависимости от задания);
- выводы, согласованные с целью работы.

1.6. Контрольные вопросы

1. Кратко опишите основные особенности языка `Python`.
2. Что такое интерпретатор? Какие интерпретаторы языка `Python` вы знаете? Чем они отличаются друг от друга?
3. Назовите основные отличия языка с динамическим от языка со статическим типизированием. К какому из этих двух видов относится `Python`?

Список литературы

1. *Rossum G.* Python Reference Manual : tech. rep. — Amsterdam, The Netherlands, 1995.

2. Maple 10 Programming Guide / M. B. Monagan, K. O. Geddes, M. K. Heal, G. Labahn, S. M. Vorkoetter, J. McCarron, P. DeMarco. — Waterloo ON, Canada : Maplesoft, 2005.
3. SciPy: Open source scientific tools for Python / E. Jones, T. Oliphant, P. Peterson, [et al.]. — 2014. — URL: http://www.scipy.org/.
4. *Маккинни У.* Python и анализ данных. — М. : ДМК Пресс, 2015. — ISBN 978-5-97060-315-4.
5. *Сламкин Б.* Секреты Python. — М. : Вильямс, 2016. — ISBN 9785845920782.
6. *Любанович Б.* Простой Python. Современный стиль программирования. — М. : Питер, 2017. — ISBN 978-5-496-02088-6.
7. Anaconda home. — 2018. — URL: https://anaconda.org/.
8. Cpython git repository. — 2018. — URL: https://github.com/python/cpython.
9. IronPython home site. — 2018. — URL: http://ironpython.net/.
10. Jython home site. — 2018. — URL: http://www.jython.org/.
11. Maple home site. — 2018. — URL: https://www.maplesoft.com/products/maple/.
12. Mathematica home site. — 2018. — URL: https://www.wolfram.com/mathematica/.
13. MatLab home site. — 2018. — URL: https://www.mathworks.com/.
14. Matplotlib home site. — 2018. — URL: https://matplotlib.org/.
15. Numba home site. — 2018. — URL: https://numba.pydata.org/.
16. NumPy home site. — 2018. — URL: http://www.numpy.org//.
17. PyPy home site. — 2018. — URL: https://pypy.org/.
18. Python home site. — 2018. — URL: https://www.python.org/.
19. SymPy home site. — 2018. — URL: http://www.sympy.org/ru/index.html.
20. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 2. Освоение основ языка Python

2.1. Цель лабораторной работы

Целью данной лабораторной работы является освоение базовых средств языка Python, изложенных в теоретической части данной главы.

2.2. Предварительные сведения

2.2.1. Основные типы данных

2.2.1.1. Числовые типы данных и элементарные вычисления

В силу динамической природы языка `Python` для создания переменной достаточно присвоить ей некоторое значение. Интерпретатор самостоятельно определит, под какой тип данных следует выделить память. Например следующий код

```
j = 1
```

создаст переменную `j` типа `int` (целый тип). Заметим, что для выяснения типа переменной можно воспользоваться встроенной функцией `type()`.

Рассмотрим несколько примеров, иллюстрирующих элементарные арифметические действия:

```
>>> 1 + 3**2 - (6 - 5)
9
>>> 0.1 + 0.2 + 0.3 + 1e4
10000.5
>>> print('Привет мир!')
```

Оператор `**` позволяет возводить в степень числа любого типа. При совершении действий над числовыми переменными различного типа происходит автоматическое приведении типа к более общему из них. Встроенная функция `print()` позволяет выводить информацию в стандартный поток вывода (консоль).

Следующий код показывает основные типы данных, доступные в Python:

```
j = 2 # Целое число
a = 5.0 # Действительное число
A = 1.0 # Регистр имеет значение
y =  1.0 + 1.0j # Комплексное число
str1 = "Это некоторый текст"
str2 = 'Одинарные кавычки тоже допустимы'
print(j, a, A, y, str1, str2)
```

2.2.1.2. Строки и их методы

Строки в `Python` являются одним из базовых типов. Они реализованы как сложные объекты, имеющие функции (методы) и поля (атрибуты). Для доступа к методам используется оператор точка `.`. Для получения полного списка атрибутов и методов можно использовать встроенную функцию `dir(object_name)`.

Рассмотрим несколько методов строк:

```
>>> msg = "Здравствуйте!"
>>> msg.upper() # в верхнем регистре
'ЗДРАВСТВУЙТЕ!'
>>> msg.lower() # в нижнем регистре
'здравствуйте!'
```

```
>>> "CAPS LOCK".lower()
'caps lock'
```

Часто приходится разбивать строки на части, используя некоторый символ в качестве разделителя. Рассмотрим как это делается:

```
animals = "Заяц,Кролик,Гусь,Селезень"
l = animals.split(',') # создаётся список из 4 строк
print(l)
```

Тройные кавычки позволяют задавать большие куски текста (длинные строки), а метод `splitlines()` позволяет разбивать такой блок текста на строки:

```
block = """1,2,32
2,76,54
3,32,454
"""
s = block.splitlines()
print(s)
['1,2,32', '2,76,54', '3,32,454']
```

Для конкатенации строк служит оператор `+`:

```
>>>"Раз" + " и " + "два!"
'Раз и два!'
```

Однако гораздо более гибок метод `format()`, который служит для форматирования строк и похож на функцию `printf` языка `C`:

```
"a = {0}, a^2 = {1}".format(a, a**2)
```

Для конвертации других типов в строку служит встроенная функция `str()`. Используя её и оператор `+`, можно формировать сложные строки, содержащие значения любых переменных. Однако предпочтительнее для таких целей использовать метод `format()`, так как он намного эффективней работает с памятью.

2.2.1.3. Логический тип и логические выражения

Зарезервированные слова `True` и `False` служат для обозначения лжи и истины:

```
X = True
Y = False
X is Y # False
X is X # True
Y is Y # True
6 < 4 # False
5 == 1 # False
```

Вхождение подстроки в строку можно проверять с помощью оператора `in`:

```
"След" in "Следствие" # True
"кость" in "плоскость" # True
```

Логические операторы конъюнкции, дизъюнкции, отрицания и строгой дизъюнкции реализованы в виде английских слов `and`, `or`, `not` и `xor`:

```
6 and 3 # True
6 or 0 # True
True is not False # True
```

Операции сравнения >, <, >= и <= можно комбинировать. Так, например, корректно следующее выражение:

```
6 <= x < 3
```

2.2.2. Управление потоком выполнения

В синтаксисе языка Python нет открывающих и закрывающих конструкций типа слов begin и end или скобок { и }. Для определения вложенности инструкций используются отступы. В качестве отступов можно применять табуляцию или четыре пробела. Официальная документация рекомендует использовать исключительно четыре пробела. Благодаря этому текст программы становится короче и более читаемым, так как программист вынужден всегда делать отступы, и иерархия управляющих конструкций всегда явно видна.

Рассмотрим оператор условного перехода if-elif-else, а также циклы for и while.

2.2.2.1. Оператор условного перехода if

Оператор условного перехода if заменяет как классический оператор if, так и оператор switch/case. Проиллюстрируем его применение:

```
a = 4
if a == 4:
    print("Четыре")
elif a == 5:
    print("Пять")
elif 0 <= a < 4:
    print("Меньше четырех")
else:
    print("Что-то иное")
>>> 'Четыре'
```

Двоеточие после каждого условия и безусловного else — обязательно. Для записи условия не требуются круглые скобки. Для повышения читаемости и наглядности можно сложные условия вычислять отдельно:

```
x = 1
y = 2
z = 3
p = 3
condition = (x < y < z < p)
if condition:
    print("Условие истинно")
else:
    print("Условие ложно")
>>> 'Условие ложно'
```

2.2.2.2. Циклы

Цикл for следует использовать для перебора элементов некоторой коллекции (множество, список, кортеж и т.д.).

Рассмотрим следующий пример:

```
for i in range(1, 10, 1):
    print(i, end=' ')
>>> '1 2 3 4 5 6 7 8 9'
```

Здесь также используется двоеточие. Встроенная функция range(start, stop, step) позволяет перебрать целые числа, начиная от start и заканчивая stop-1, с шагом step. Заметим также, что правая граница диапазона не входит в конечный набор чисел. Это поведение в работе с индексами характерно для языка Python и будет встречаться далее повсеместно.

В языке Python цикл while имеет стандартный синтаксис, поэтому мы ограничимся лишь одним примером:

```
x = 1.3
while x > 1:
    x = x - 1.1
```

2.3. Списки

В языке Python нет встроенных классических массивов, однако есть более гибкий тип данных — список (list). Список может содержать элементы различных типов, обеспечивает доступ к элементам по индексам, даёт возможность добавлять и удалять элементы. Список Python универсален и может выполнять роль стека, очереди и двусторонней очереди.

Классические массивы отсутствуют в Python, однако этот недостаток устраняется библиотекой NumPy, которую очень кратко будет рассмотрена позже.

2.3.1. Создание списка

Создаётся список посредством квадратных скобок [] или функции list():

```
>>> l = [] # Пустой список
>>> l = [6, 7, 8, 9, 10]
>>> print(l)
[6, 7, 8, 9, 10]
>>> [False, 1, "Привет"] # Список с элементами разного типа
>>> A = ['а','б','в']
>>> print("A:", A, " Количество элементов:", len(A))
"A: ['а','б','в']  Количество элементов: 3"
```

Встроенная функция len универсальна и может использоваться для вычисления длины любых коллекций. В нашем примере с помощью этой функции возвращается количество элементов в списке.

Оператор сложения + позволяет объединить два списка в один, а не производит поэлементного сложения:

```
>>> a = [1, 2, 3, 4, 5]
>>> b = [5, 4, 3, 2, 1]
>>> a + b
[1, 2, 3, 4, 5, 5, 4, 3, 2, 1]
```

К элементам списка можно обращаться по индексу, начиная с 0. Поддерживаются также вырезки и сечения, то есть извлечения определённого диапазона элементов, для этого используется синтаксис start:stop:step. Следует иметь ввиду, что правая граница никогда не включается в результат:

```
>>> b[0:3]
[5, 4, 3]
>>> b[:3]
[5, 4, 3]
>>> b[0:4:2]
[5, 3]
>> b[::2] # каждый второй
[5, 3, 1]
```

Список имеет одномерную структуру, однако можно вкладывать списки друг в друга (полноценные многомерные массивы доступны при использовании библиотеки NumPy):

```
l = [[11, 12], [21, 22]]
l[0][1] # 12
```

2.3.2. Методы списков

Список является объектом, поэтому у него есть различные методы. Оператор точка `.` используется для доступа к методам и атрибутам. Метод `append` добавляет элемент в конец списка, метод `pop()` удаляет и возвращает последний элемент списка, метод `sort` сортирует список по возрастанию или убыванию (если задан параметр `reverse`):

```
>>> l = [1, 2, -1, 4, 5, 6]
>>> l.append(7)
[1, 2, -1, 4, 5, 6, 7]
>>> l.pop()
[1, 2, -1, 4, 5, 6]
>>> l.sort(); l
[-1, 1, 2, 4, 5, 6]
```

Все доступные для использования методы можно посмотреть в документации языка или же при использование оболочки Jupyter Notebook, нажав клавишу Tab после поставленной точки. Справку по конкретному методу можно получить, нажав Shift-Tab после имени метода.

2.3.3. Конструирование списков с помощью списковых сборок

Python позволяет конструировать списки гибким способом, называемым *списковым включением* или *списковой сборкой* (list comprehension).

Начнём рассмотрение с примера, в котором сгенерируем список, содержащий последовательность целых чисел от 1 до 10. Сделаем это с помощью цикла `for`. Создадим пустой список, а затем в цикле заполним его значениями:

```
l = []
for i in range(1, 11, 1):
    l.append(i)
print(l)
>>> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

То же самое можно сделать более коротким способом, если воспользоваться специальным синтаксисом списковой сборки. Фактически цикл будет записан внутри квадратных скобок:

```
l = [i for i in range(1, 11, 1)]
print(l)
>>> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

В нашем простом примере можно сконструировать список намного короче, использовав конструктор `list`:

```
l = list(range(1, 11, 1))
print(l)
>>> [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Усложним наш пример, сформировав список квадратов целых чисел. Воспользуемся сначала циклом:

```
l = []
for i in range(1, 11, 1):
    l.append(i**2)
print(l)
>>>[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Затем сделаем то же самое с помощью списковой сборки:

```
l = [i**2 for i in range(1, 11, 1)]
print(l)
>>>[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Подобное можно сделать и с помощью конструктора `list`, но на этот раз запись не будет короче:

```
l = list(i**2 for i in range(1, 11, 1))
print(l)
>>>[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Продолжим усложнять наш пример, вычисляя квадраты только чётных чисел от 1 до 10. Для этого придётся использовать условный оператор `if` для выбора только тех элементов, которые делятся на 2 без остатка:

```
l = []
for i in range(1, 11, 1):
    # оператор % позволяет найти остаток от деления
    if i % 2 == 0:
        l.append(i**2)
print(l)
>>> [4, 16, 36, 64, 100]
```

То же самое можно сделать с помощью списковой сборки, но заметно короче:

```
l = [i**2 for i in range(1, 11, 1) if i % 2 == 0]
print(l)
```

Сформируем теперь таблицу умножения, воспользовавшись вложенными списками. Сначала сделаем это с помощью цикла, а потом с помощь списковой сборки:

```
l = []
for i in range(1, 10, 1):
    M = []
    for j in range(1, 10, 1):
        M.append(i * j)
    l.append(M)
l = [[i * j for j in range(1, 10, 1)] for i in range(1, 10, 1)]
```

Добавим теперь условие на чётность и сформируем таблицу умножения только чётных чисел:

```
l = []
for i in range(1, 10, 1):
    if i % 2 == 0:
        M = []
        for j in range(1, 10, 1):
            if j % 2 == 0:
                M.append(i * j)
        l.append(M)
l = [[i * j for j in range(1, 10, 1) if j % 2 == 0] for i in
 ↪  range(1, 10, 1) if i % 2 == 0]
```

2.3.4. Использование функции zip

Параллельный перебор элементов из двух и более списков можно записать короче, если воспользоваться встроенной функцией `zip`, которая последовательно извлекает из переданных ей списков по элементу и формирует их в кортежи.

Пусть у нас есть список координат `X` и `Y`, из которых надо сформировать список точек, в котором каждая точка представлена парой `(x, y)` (кортеж). Рассмотрим сначала, как это сделать в цикле, а затем в списковой сборке:

```
X = [1, 2, 3, 4, 5, 6]
Y = [1, 2, 3, 4, 5, 6]
Ps = []
for x, y in zip(X, Y):
    # Обратите внимание на двойные скобки
    print(x, y)
    Ps.append((x, y))
print(Ps)
>>> [(1, 1), (2, 2), (3, 3), (4, 4), (5, 5), (6, 6)]
Ps = [P for P in zip(X, Y)]
print(Ps)
# Иначе
Ps = list(zip(X, Y))
print(Ps)
```

Важно помнить, что функция `zip()` создаёт именно *итератор*, а не список кортежей из переданных ей списков, а значит результатом её работы можно воспользоваться только один раз.

Рассмотрим пример:

```
z = zip(r, r)
print("Тип объекта, возвращаемого функцией zip():", type(z))
print("Проходим по элементам z первый раз")
for item in z:
    print(item, end=' ')
# Это итератор, поэтому после использования в цикле память
 ↪  освободилась!
# Повторим еще раз и убедимся, что в z теперь пусто!
print("\nПроходим по элементам z второй раз")
for item in z:
    print(item)
```

Чтобы из итератора сделать список, можно воспользоваться функцией `list()`.

2.4. Кортежи

В Python определён тип данных, называемый *кортежом* (tuple) и представляющий собой упорядоченный набор фиксированной длины. Кортеж является иммутабельным (immutable) объектом, что означает невозможность модификации уже созданного объекта. В случае кортежа это выражается в невозможности добавлять новые элементы, заменять старые или удалять уже имеющиеся. Однако возможно соединить два кортежа в один с помощью оператора `+`, создав новый кортеж:

```
>>> l = (1, 2, 3, 4)
>>> l[2] # 3
>>> for item in l:
        print(item, end=' ')
1 2 3 4
# l[0] = 1 # недопустимая операция!
```

Для создания кортежа можно использовать встроенную функцию `tuple` и синтаксис списковых сборок:

```
>>> t = tuple(i**2 for i in range(11))
# (0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100)
```

В кортеже удобно хранить данные, которые не предполагается изменять. Например, это могут быть координаты точки или вектора.

С понятием кортежа связанна операция распаковки переменных. Проиллюстрируем её примером:

```
>>> x, y = (1, 2)
# в результате x = 1, y = 2
>>> (x, y) = [1, 2] # круглые скобки не обязательны
# аналогично x = 1, y = 2
>>> x, y, z, = [1, 2, 3, 4, 5, 6]
# x = 1, y = 2, z = 3
```

Обратите внимание на запятую в последнем примере после переменной `z`. Она необходима, если количество элементов в наборе справа больше числа переменных слева.

Кортежи встречаются в языке Python повсеместно. Мы уже сталкивались с кортежами и распаковкой переменных, когда использовали функцию `zip` для одновременного интегрирования по двум и более массивам. Кроме того, они часто применяются в функциях в случае, если необходимо вернуть несколько значений.

2.5. Словари

Удобный тип данных для задания составных структур — словарь (ассоциативный массив, отображение (map)). Словарь представляет собой расширенный вариант массива, где элементы индексируются не просто числами от 0 до N, а ключами общего вида, которые могут иметь любой базовый тип, однако чаще всего используют строки для придания ключам осмысленных названий.

2.5.1. Создание словаря. Методы словаря

Создать пустой словарь можно несколькими способами:

```
# Пустой словарь
D = {}
print(type(D))
# или используем конструктор
D = dict()
print(type(D))
```

Для задания содержимого словаря используется синтаксис `key : value`:

```
D = {
    'Ключ 1': 'Значение 1',
    'Ключ 2': 'Значение 2',
    'Ключ 3': 'Значение 3'
}
```

Можно также использовать конструктор `dict`:

```
keys = ['раз', 'два', 'три']
values = [1, 2, 3]
D2 = dict(zip(keys, values))
```

Доступ к элементам словаря осуществляется путём указания ключа:

```
D['Ключ 1']
```

Попытка обратиться к несуществующему ключу не методом `get` вызовет ошибку.

Для добавления новых элементов достаточно указать желаемый ключ и присвоить ему соответствующее значение:

```
D['Ключ 4'] = [1, 2, 3, 4]
print(D)
# {'Ключ 1': 'Значение 1', 'Ключ 2': 'Значение 2', 'Ключ 3':
 ↪  'Значение 3', 'Ключ 4': [1, 2, 3, 4]}
```

Операция + не работает со словарями, но для объединения двух словарей можно использовать метод `update`:

```
new_D = {
    'Ключ 05': ('a', 'b'),
    'Ключ 1': 'абвгд'
}
D.update(new_D)
print(D)
# {'Ключ 1': 'абвгд', 'Ключ 2': 'Значение 2', 'Ключ 3':
 ↪  'Значение 3', 'Ключ 4': [1, 2, 3, 4], 'Ключ 05': ('a', 'b')}
```

Важно помнить, что сохранение порядка ключей в словаре не гарантируется. В нашем примере порядок сохраняется, но при большем количестве элементов он может нарушится. Если порядок ключей важен, то следует воспользоваться типом данных `OrderedDict` из модуля `collections`.

2.5.2. Перебор значений и ключей в цикле

Для перебора элементов словаря применяет цикл `for` и несколько методов словаря:

```
# перебор всех ключей
for key in D:
    print(key)

# перебор всех значений
for value in D.values():
    print(value)

# перебор пар (ключ, значение)
for key, value in D.items():
    print("{0} <=> {1}".format(key, value))
```

Для проверки, существует ли тот или иной ключ в словаре, можно использовать метод `get`:

```
if D.get('Ключ 4') is None:
    print('Ключа № 4 не существует')
else:
    print('Ключ 4 существует')
```

2.6. Функции

Функции являются базовым средством для разделения кода сложной программы на независимые, логически обоснованные части. Функции в Python являются объектом, поэтому ими можно манипулировать так же, как и другими объектами: присваивать переменным, передавать в функции в виде аргумента, помещать в списки в виде элементов и т.д.

2.6.1. Задание функции

Для задания функций используется инструкция `def`, сразу после которой необходимо указать желаемое имя функции.

Рассмотрим ряд примеров, показывающих задание простейших функций:

```
def func_01():
    """Строка документации"""
    print('Hello world!')

def func_02(x):
    """Нахождение квадрата числа"""
    return x**2

def func_03(x, y): return (x**2, y**2)
```

Первая функция не принимает никаких аргументов и не возвращает никакие значения — она лишь распечатывает сообщение «Hello World!» в консоль. Вторая функция принимает один аргумент и возвращает значение этого аргумента, возведённое в квадрат. Третья функция принимает два аргумента и возвращает кортеж из двух квадратов своих аргументов.

Если тело функции состоит из одной строки, то можно сэкономить место, записав строку сразу после двоеточия. Также желательно каждую функцию сопровождать строкой с пояснением, которая разъясняет назначение функции, возможные типы её аргументов, возвращаемое значение, показывает примеры её использования и т.д.

Типы аргументов функции не фиксируются. Можно передавать в качестве аргумента любой объект (duck typing). Однако в теле функции можно предусмотреть средства проверки типов аргументов, а также прописать в документации, с какими аргументами функция может работать:

```
def func_03(n):
    if not isinstance(n, int):
        print('Аргументы функции должны быть целым!')
        return None
    else:
        return n * (n - 1)
```

Однако следует стараться сделать функции более стабильными и попытаться сконвертировать аргумент в нужный тип автоматически:

```
def func_03(n):
    return int(n) * (int(n) - 1)
```

2.6.2. Аргументы функции

Аргументам функции можно присваивать значения по умолчанию. При этом такие аргументы становятся опциональными и их можно не указывать при вызове функции:

```
>>> def func_04(x, y=1):
>>>     return (x**2, y**3)
>>> func_04(2)
(4, 1)
>>> func_04(1, 4)
(1, 64)
```

Чтобы не запоминать порядок следования аргументов, можно указывать их вместе с названиями (keywords arguments):

```
func_04(2, y=2), func_04(x=2, y=2)
# ((4, 8), (4, 8))
```

Два специальных оператора `*` и `**` позволяют передавать внутрь функции произвольные список и словарь с аргументами. В данном примере переменная `arguments` — это список со значениями, которые при распаковке будут переданы в функцию в качестве первого, второго, третьего и т.д. аргументов. Переменная `keywords` является словарём, и при использовании оператора `**` в функцию будут переданы аргументы по ключевым словам. Внутри функции их можно разобрать в цикле и обработать:

```
def func_05(*arguments, **keywords):
    for arg in arguments:
        print(arg)
    for (key, value) in keywords.items():
        print(key, ":", value)
func_05(1, 2, 3, x=4, y=5, z=6, p=7)
```

```
1
2
3
x : 4
y : 5
z : 6
p : 7
```

Используя оператор `*`, можно распаковать готовый список внутри функции:

```
def func_06(x, y, z):
    return x + y + z
func_06(*[1, 2, 3])
# 6
```

Аналогично, используя оператор `**`, можно распаковать словарь. Это особенно удобно, если функция принимает множество необязательных аргументов, которые можно заранее организовать в виде словаря, а потом передать в функцию для обработки:

```
def func_07(n=1, string='слово'):
    return n * string
args = {'n': 5, 'string': 'текст '}
func_07(**args)
# 'текст текст текст текст текст '
```

2.7. Задание

1. Повторите примеры из раздела 2.2.1, используя интерпретатор Python в интерактивном режиме.
2. Выполните задания из раздела 2.8.

2.8. Последовательность выполнения работы

Для выполнения данной работы вы можете как использовать интерактивную оболочку Jupyter, так и создавать программы в виде отдельных файлов.

1. Последовательно возводите число 2 в возрастающие натуральные степени. В какой момент возникнет переполнение точности?
2. Как наиболее ёмко на языке Python записать следующее математическое условие: $x \in (-100, -20) \cup [0, 20) \cup (30, 40]$?

3. Замените следующий цикл списковой сборкой:
```
l = []
for n in range(0, 20, 1):
  if n % 3 != 1:
    l.append(n)
```
4. Пусть дан список с текстовыми элементами следующего вида:
```
['1', '2', '3', '4', '5']
```
Преобразуйте его в список с элементами целого типа.
5. Пусть дан список из 10 элементов. Как с помощью срезов получить элементы с 5 по 10, все элементы с нечётным индексом, все элементы с чётным индексом?
6. Как с помощью кортежа можно обменять (swap) значения переменных? Например, если $x = 1$ и $y = 2$, то после обмена $x = 2$, а $y = 1$.
7. К элементам кортежа можно обращаться по индексу. Можно ли заменить значения конкретного элемента, обратившись к нему по индексу? Например:
```
c = (1, 2, 3, 4, 5)
c[2] = 0
```
8. Взяв любой словарь из раздела 2.5, поменяйте его ключи и значения местами.
9. Измените следующую функцию так, чтобы все её аргументы стали необязательными:
```
def func_04(x, y=1):
  return (x**2, y**3)
```
В качестве справочной литературы рекомендуется использовать следующие источники [1—3]

2.9. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:
- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку цели работы;
- описание результатов выполнения задания:
  - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
  - листинги (исходный код) программ (если они есть);
  - результаты выполнения программ (текст или снимок экрана в зависимости от задания);
- выводы, согласованные с целью работы.

2.10. Контрольные вопросы

1. Какие основные типы данных языка `Python` вы знаете?
2. В чем разница между списком (`list`) и кортежом (`tuple`) в языке `Python`?
3. Какие инструкции управления потоком выполнения есть в языке `Python`?
4. Что такое списковые сборки и для чего они предназначены?
5. Для чего нужны генераторы? Не дублируют ли они функционал списков?
6. Можно ли обратиться к символу строки по индексу? Можно ли заменить произвольный символ у уже существующей строки?
7. Можно ли обратиться к элементу кортежа по индексу? Можно ли заменить произвольный элемент кортежа?
8. Могут ли списки `Python` хранить данные произвольного типа?
9. Какими методами обладает список в языке `Python`?

10. Какую инструкцию следует использовать для сравнения двух переменных булева типа?
11. Какую инструкцию следует использовать для проверки вхождения подстроки в строку?
12. Каким образом можно последовательно перебрать все элементы списка? Можно ли похожим образом поступить с кортежем? Со строкой?
13. Какие функции создают список, кортеж, словарь?
14. Какие методы есть у строки?
15. Что такое словарь в языке `Python`? Какие у него есть аналоги в других языках программирования?
16. Как упорядочены элементы словаря?
17. Как можно последовательно перебрать все элементы словаря, не зная его ключей?
18. Как объявляется функция в языке `Python`?
19. Как задать обязательные аргументы функции? Необязательные аргументы?
20. Для чего служат синтаксические конструкции `*args` и `**kwargs`? Приведите пример.
21. Можно ли передавать в функцию другую функцию?
22. Можно ли возвращать из функции другую функцию? Проверьте на примере.
23. Какая специфика есть у списков при передаче их внутрь функций?
24. Используя документацию языка `Python`, ответьте на следующие вопросы:
    - Для чего нужны функции `map`, `filter` и `reduce`?
    - Для чего можно использовать встроенные функции `any` и `all`?
    - Как осуществить вывод в консоль без перевода на новую строку в конце вывода?

Список литературы

1. *Любанович Б.* Простой Python. Современный стиль программирования. — М. : Питер, 2017. — ISBN 978-5-496-02088-6.
2. Python home site. — 2018. — URL: `https://www.python.org/`.
3. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 3. Создание массивов NumPy

3.1. Цель лабораторной работы

Целью данной лабораторной работы является знакомство с основными средствами создания массивов библиотеки NumPy.

3.2. Предварительные сведения

3.2.1. Библиотека `NumPy`

3.2.1.1. Общая характеристика библиотеки `NumPy`

Библиотека `NumPy` (Numerical Python) привносит в `Python` новый тип данных, который предназначен для хранения и работы с массивами числовых данных, а также реализует большое количество разнообразных функций для манипуляции, генерации, фильтрации, сортировки массивов. Реализованы также разнообразные математические, в том числе используются функции-обёртки процедур библиотеки LAPACK, позволяющие решать задачи линейной алгебры.

В случае если вы используете WinPython или Anaconda `Python`, то библиотека `NumPy` уже установлена. В случае Miniconda её надо установить, выполнив команду `conda install numpy`

После установки можно будет импортировать `NumPy`.

Подключение любых библиотек осуществляется командой `import`. Инструкция `as` позволяет переименовать импортируемый модуль, чтобы название было короче и им было удобнее пользоваться при наборе кода. Для библиотеки `NumPy` стандартом стало двухбуквенное сокращение `np`:

```
import numpy as np
```

После импорта все функции и подмодули библиотеки `NumPy` станут доступны посредством оператора точка `'.'`. Подобная изоляция *пространства имён* разных модулей позволяет избежать конфликта имён и служит инструментом структурирования функционала библиотеки, позволяя распределить функции по тематическим подмодулям.

Массивы `NumPy` в первом приближении похожи на встроенный в `Python` тип данных `list`, но работают гораздо быстрее, так как каждый массив хранит данные только одного типа, и их размер задаётся при инициализации. Это позволяет сократить накладные расходы памяти и уменьшить время доступа. Большинство функций и структур данных библиотеки `NumPy` реализованы на языках C/C++ и Fortran. Также активно используется векторизация безындексных операций над массивами, позволяющая задействовать все ядра многоядерных процессоров и распределить работу между ними.

В данном разделе даётся краткое введение в основные возможности `NumPy` версии не ниже `1.13`.

3.2.2. Создание NumPy массива. Основные операции

3.2.2.1. Создание массива

Основной функцией для создания массива служит `np.array()`. Она принимает перечисляемые типы данных (`list`, `tuple`, генераторы и т.д.) и создаёт на их основе

numpy-массив. Переданная структура должна состоять из элементов одного типа. В случае числовых данных допускается комбинация целых, рациональных, вещественных и комплексных числе, однако при создании массива они будут приведены к наиболее общему типу.

Рассмотрим ряд примеров:

```
a = np.array([1, 2, 3, 4, 5])
print(a, 'тип элементов', a.dtype)
# если хотя бы один элемент вещественный, то весь массив будет
 ↪  иметь вещественный тип
a = np.array([1, 2.0, 3, 4, 5])
print(a, 'тип элементов', a.dtype)
# то же самое справедливо для комплексного типа
a = np.array([1, 2.0, 3 + 1j, 4, 5])
print(a, 'тип элементов', a.dtype)
# можно передавать строки, тогда результатом будет символьный
 ↪  массив
a = np.array([1, 2.0, 3, '4', 5])
print(a, 'тип элементов', a.dtype)
a = np.array([1, 2.0, 3 + 1j, 'a', 5])
print(a, 'тип элементов', a.dtype)
# Программа даст следующий вывод
# [1 2 3 4 5] тип элементов int64
# [ 1.  2.  3.  4.  5.] тип элементов float64
# [ 1.+0.j  2.+0.j  3.+1.j  4.+0.j  5.+0.j] тип элементов
 ↪  complex128
# ['1' '2.0' '3' '4' '5'] тип элементов <U32
# ['1' '2.0' '(3+1j)' 'a' '5'] тип элементов <U64
```

Тип передаваемых элементов можно указать явным образом с помощью аргумента dtype:

```
a = np.array([1, 2, 3, 4, 5], dtype=np.float64)
print(a, 'тип элементов', a.dtype)
# [ 1.  2.  3.  4.  5.] тип элементов float64
```

Numpy поддерживает стандартные типы данных Python, а также определяет более специфические численные типы, например, беззнаковые целые (uint8, uint16, uint32 и uint64), действительные числа (float16, float32 и float64), комплексные числа (complex64, complex128).

Для создания многомерных массивов можно использовать вложенные списки и списковые сборки:

```
a = np.array([[i+j for i in range(5)] for j in range(5)])
print(a)
#[[0 1 2 3 4]
# [1 2 3 4 5]
# [2 3 4 5 6]
# [3 4 5 6 7]
# [4 5 6 7 8]]
```

Также имеется ряд функций, предназначенных для создания массивов из нулей, единиц, одинаковых элементов или просто для выделения памяти:

```
# Массив, состоящий целиком из нулей
a = np.zeros(5, dtype=int)
b = np.zeros(shape=(5, 2), dtype=int)
# Массив, состоящий целиком из единиц
a = np.ones(shape=(2, 2))
```

```
# Единичная матрица 5x5
E = np.eye(5)
```

Функция `np.arange(start, stop, step)` создаёт массив из элементов арифметической прогрессии, начиная от `start` и заканчивая `stop` (но не включая его) с шагом `step`. Последовательность может состоять из вещественных чисел:

```
a = np.arange(1, 6, 2)
b = np.arange(1.0, 6.0, 2.0)
a, a.dtype, b, b.dtype
```

Функция `np.linspace(a, b, N)` разбивает отрезок (a, b) на N частей. Эту функцию удобно применять для задания области определения кривых:

```
l = np.linspace(0, 1, 20)
```

Наконец, если необходимо лишь выделить память под данные определённого типа, то можно воспользоваться функцией `np.empty`. Значениями будут произвольные, случайно оказавшиеся в соответствующих ячейках памяти данные:

```
a = np.empty(shape=(2, 2), dtype=np.float64)
```

3.2.2.2. Характеристики массива и доступ к элементам

У каждого `numpy`-массива имеется ряд характеристик, которые хранятся в виде атрибутов:

- число измерений (целое число, атрибут `ndim`);
- форма (или иначе число) элементов по каждому измерению (кортеж из целых чисел, атрибут `shape`);
- общий размер массива, равный суммарному количеству элементов, содержащихся в массиве (целое число, атрибут `size`);
- тип данных массива (атрибут `dtype`).

Примеры:

```
a = np.empty(shape=1)
b = np.empty(shape=(2, 3))
c = np.empty(shape=(2, 3, 4), dtype=np.int64)
print('a: ndim={0}, shape={1}, size={2},
 ↪  dtype={3}'.format(a.ndim, a.shape, a.size, a.dtype))
print('b: ndim={0}, shape={1}, size={2},
 ↪  dtype={3}'.format(b.ndim, b.shape, b.size, b.dtype))
print('c: ndim={0}, shape={1}, size={2},
 ↪  dtype={3}'.format(c.ndim, c.shape, c.size, c.dtype))
a: ndim=1, shape=(1,), size=1, dtype=float64
b: ndim=2, shape=(2, 3), size=6, dtype=float64
c: ndim=3, shape=(2, 3, 4), size=24, dtype=int64
```

Стандартным способом доступа к элементам массива является доступ по индексам. Здесь синтаксис сходен со стандартным синтаксисом языка `Python` и с другими языками, где индексация начинается с нуля. Кроме доступа к конкретным элементам, можно использовать синтаксис срезов, получая доступ сразу к набору элементов с заданным шагом. Точно также, как и в случае списков, последний элемент диапазона среза не входит в результат среза:

```
a = np.array([[11, 12, 13], [21, 22, 23], [31, 32, 33]])
# Вся вторая строка
print(a[1, :])
# Весь первый столбец
print(a[:, 0])
# [21 22 23]
# [11 21 31]
```

Перебрать все элементы можно с помощью цикла `for`. Обход при этом будет производиться по строкам:

```
for row in a:
    print('Строка', row, end=' и ее элементы: ')
    for item in row:
        print(item, end=' ')
    print()
# Строка [11 12 13] и ее элементы: 11 12 13
# Строка [21 22 23] и ее элементы: 21 22 23
# Строка [31 32 33] и ее элементы: 31 32 33
```

Функция `np.ndenumerate` является эквивалентом функции `enumerate` и служит для последовательного перебора всех элементов массива, возвращая кортеж `(index, item)`, где `index` в свою очередь является кортежем индексов элемента `item`.

Рассмотрим пример:

```
for ((i, j), item) in np.ndenumerate(a):
    print('a[{0}, {1}] = {2}'.format(i, j, item))
# a[0, 0] = 11
# a[0, 1] = 12
# a[0, 2] = 13
# a[1, 0] = 21
# a[1, 1] = 22
# a[1, 2] = 23
# a[2, 0] = 31
# a[2, 1] = 32
# a[2, 2] = 33
```

Обратите внимание, что и здесь обход производится по строкам. Это связано со способом хранения элементов массива в ячейках памяти. Так как структуры данных `NumPy` реализованы на языке `C`, то элементы многомерных массивов располагаются в памяти по строкам, поэтому перебор по строкам — это наиболее быстрый способ последовательного их перебора.

Кратко перечислим основные функции, которые служат для изменения формы массивов:

- `np.reshape` позволяет переформатировать массив;
- `np.concatenate`, `np.vstack`, `np.hstack` позволяют разными способами слить несколько массивов в один;
- `np.split`, `np.split`, `np.hsplit` разбивают массив на части.

3.2.2.3. Алгебраические действия и математические функции

Стандартные циклы `Python` весьма медленные. Это обусловлено динамической природой языка, так как при каждой операции над элементами списка интерпретатор должен проверить тип элемента, который не известен заранее. Библиотека `NumPy` устраняет этот недостаток с помощью *векторизации* операций. Векторизованные операции в библиотеке `NumPy` реализованы посредством *универсальных функций* (ufuncs), главная задача которых состоит в быстром выполнении повторяющихся операций над значениями из массивов библиотеки NumPy.

При использовании больших массивов следует избегать обращения к элементам по индексам и стараться работать с массивами в безындексной форме. Например, если необходимо сложить, умножить, вычесть, разделить или возвести в степень все элементы массива, то можно воспользоваться обычными операторами `+`, `*`, `-`, `/` и `**`, как показано в примере:

```
x = np.random.random(size=10**6)
y = np.random.random(size=10**6)
%timeit x + y
%timeit x * y
%timeit x - y
%timeit x / y
%timeit x**2
```

```
2.25 ms ± 16.1 µs per loop (mean ± std. dev. of 7 runs, 100
 ↪  loops each)
2.24 ms ± 8.93 µs per loop (mean ± std. dev. of 7 runs, 100
 ↪  loops each)
2.25 ms ± 9.63 µs per loop (mean ± std. dev. of 7 runs, 100
 ↪  loops each)
4.27 ms ± 2.56 µs per loop (mean ± std. dev. of 7 runs, 100
 ↪  loops each)
1.52 ms ± 4.75 µs per loop (mean ± std. dev. of 7 runs, 1000
 ↪  loops each)
```

Если же теперь проделать все те же действия с помощью обычного цикла, обращаясь к элементам массива по индексу, то суммирование будет произведено более чем в 100 раз медленнее:

```
%%timeit
for i in range(10**6):
    x[i] = x[i] + y[i]
```

```
565 ms ± 2.73 ms per loop (mean ± std. dev. of 7 runs, 1 loop
 ↪  each)
```

Кроме арифметических операторов определены универсальные функции для стандартных математических действий, например для нахождения абсолютного значения всех элементов, среднего, максимального и минимального значений. Также векторизированы элементарные математически функции (тригонометрические, гиперболические, экспонента, логарифмы, квадратный корень).

Рассмотрим примеры и оценим ускорение операций:

```
%timeit [math.sin(item) for item in x]
%timeit np.sin(x)
```

```
327 ms ± 290 µs per loop (mean ± std. dev. of 7 runs, 1 loop
 ↪  each)
45.6 ms ± 43.5 µs per loop (mean ± std. dev. of 7 runs, 10 loops
 ↪  each)
```

```
%timeit [math.sqrt(item) for item in x]
%timeit np.sqrt(x)
```

```
278 ms ± 280 µs per loop (mean ± std. dev. of 7 runs, 1 loop
 ↪  each)
4.26 ms ± 406 ns per loop (mean ± std. dev. of 7 runs, 100 loops
 ↪  each)
```

Как видно из результатов, векторизированные функции обеспечивают ускорение на порядок, а в ряде случаев и на два порядка. Однако встроенные функции выигрывают в случае операций над скалярными данными:

```
%timeit math.sin(2.0)
%timeit np.sin(2.0)
```

```
171 ns ± 0.719 ns per loop (mean ± std. dev. of 7 runs, 10000000
 ↪  loops each)
1.15 µs ± 0.977 ns per loop (mean ± std. dev. of 7 runs, 1000000
 ↪  loops each)
```

```
%timeit math.sqrt(2.0)
%timeit np.sqrt(2.0)
```

```
149 ns ± 0.684 ns per loop (mean ± std. dev. of 7 runs, 10000000
 ↪  loops each)
1.14 µs ± 3.53 ns per loop (mean ± std. dev. of 7 runs, 1000000
 ↪  loops each)
```

Некоторые из векторизированных функций могут задействовать несколько ядер процессора. Такова, например, функция `dot`, вычисляющая скалярное произведение двух массивов:

```
%timeit np.dot(x, y)
```

```
956 µs ± 15 µs per loop (mean ± std. dev. of 7 runs, 1000 loops
 ↪  each)
```

3.3. Задание

1. С помощью библиотеки `NumPy` [2] выполните действия по созданию массива и выполнению операций с массивами (см. раздел 3.4).

3.4. Последовательность выполнения работы

1. Создайте массив, содержащий последовательность чисел от 1 до 100. Какую функции из библиотеки `NumPy` лучше всего использовать для этой цели?
2. Чем отличается функция `np.eye` от функции `np.identity`? Сравните результаты вызовов `np.eye(3, 3)` и `np.identity(3)`.
3. Используя функции `np.linspace` и `np.sin`, вычислите значения функции $\sin(x)$ в 100 точках отрезка $[-\pi, \pi]$.
4. Найдите число измерений, число элементов по каждому измерению, общее число элементов и тип элементов для следующего массива:
   `A = np.random.random((3, 2, 1))`
5. Как с помощью синтаксиса срезов получить все элементы первой строки двумерного массива?
6. Создайте двумерный массив, каждый элемент которого равен сумме своих индексов. Используйте для этого функцию `np.ndenumerate`.

В качестве справочной литературы рекомендуется использовать следующие источники [1; 3; 4].

3.5. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:

- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку цели работы;
- описание результатов выполнения задания:
  - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;

  - листинги (исходный код) программ (если они есть);
  - результаты выполнения программ (текст или снимок экрана в зависимости от задания);
- выводы, согласованные с целью работы.

3.6. Контрольные вопросы

Под термином *массив* в вопросах всегда подразумевается массив библиотеки `NumPy`.

1. Дайте краткую характеристику библиотеке `NumPy`.
2. Какое стандартное сокращение принято использовать при импорте библиотеки `NumPy`?
3. Какие недостатки встроенного в `Python` типа `list` призван устранить массив `NumPy`?
4. Какие типы данных поддерживает `ndarray`?
5. Какие функции для создания массивов вы знаете?
6. В каких случаях стоит предпочесть `ndarray` встроенному списку `Python`?
7. Как создать многомерный массив?
8. Массивы какого вида можно создавать с помощью специальных функций `eye`, `ones`, `zeros` и `empty`?
9. Как явно указать тип элементов массива?

Список литературы

1. *Любанович Б.* Простой Python. Современный стиль программирования. — М. : Питер, 2017. — ISBN 978-5-496-02088-6.
2. NumPy home site. — 2018. — URL: `http://www.numpy.org//`.
3. Python home site. — 2018. — URL: `https://www.python.org/`.
4. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 4. Манипуляции с массивами NumPy

4.1. Цель лабораторной работы

Целью данной лабораторной работы является знакомство с основными средствами векторных вычислений с использованием массивов `NumPy`.

4.2. Предварительные сведения

4.2.1. Анализ массивов NumPy

4.2.1.1. Статистические характеристики данных

В библиотеке `NumPy` определены функции для вычисления статистических характеристик данных, содержащихся в массиве. Все эти функции могут применяться как ко всем элементам, так и к элементам лишь вдоль указанной размерности. Перечислим важнейшие из этих функций:

- `np.max`, `np.min` — максимум и минимум;
- `np.sum`, `np.cumsum` — сумма и накопленная (кумулятивная) сумма;
- `np.prod`, `np.cumprod` — произведение или накопленное (кумулятивное) произведение;
- `np.mean`, `np.var` — среднее и вариация (эмпирическая дисперсия);
- `np.percentile` — квантили.

Рассмотрим примеры применения этих функций.

```
a = np.arange(1, 6, 1)
res = """a = {},
max(a) = {}, min(a) = {},
sum(a) = {}, cumsum(a) = {},
prod(a) = {}, cumprod(a) = {},
mean(a) = {}, var(a) = {},
percentile(a, q=50) = {} # медиана
""".format(a, np.max(a), np.min(a), np.sum(a), np.cumsum(a),
 ↪  np.prod(a),
           np.cumprod(a),np.mean(a), np.var(a), np.percentile(a,
            ↪  q=50))
print(res)
```

```
a = [1 2 3 4 5],
max(a) = 5, min(a) = 1,
sum(a) = 15, cumsum(a) = [ 1  3  6 10 15],
prod(a) = 120, cumprod(a) = [  1   2   6  24 120],
mean(a) = 3.0, var(a) = 2.0,
percentile(a, q=50) = 3.0 # медиана
```

```
b = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# суммирование по первому и второму измерениям, а также по всем
 ↪  элементам
b.sum(axis=0), b.sum(axis=1), b.sum()
```

```
(array([12, 15, 18]), array([ 6, 15, 24]), 45)
```

4.2.1.2. Логические операции над массивами

Операции сравнения (<, > и т.д.) реализованы в библиотеке `NumPy` как поэлементные векторизованные функции. Пользоваться ими можно также, как и арифметическими операторами. В качестве результата будут возвращены «логические маски»:

```
a = np.array([1, -1, 2, 0, 3, -2, 1])
print('a > 0', a > 0)
print('a < 0', a < 0)
print('a >= 0', a >= 0)
print('a <= 0', a <= 0)
print('a == 0', a == 0)
```

```
a > 0 [ True False  True False  True False  True]
a < 0 [False  True False False False  True False]
a >= 0 [ True False  True  True  True False  True]
a <= 0 [False  True False  True False  True False]
a == 0 [False False False  True False False False]
```

Если передать результаты операции логического сравнения массиву в качестве индексов, то будут отобраны только те элементы, которые удовлетворяют переданному условию:

```
print('Элементы a > 0:', a[a > 0])
print('Элементы a < 0:', a[a < 0])
print('Элементы a >= 0:', a[a >= 0])
print('Элементы a <= 0:', a[a <= 0])
print('Элементы a == 0:', a[a == 0])
```

```
Элементы a > 0: [1 2 3 1]
Элементы a < 0: [-1 -2]
Элементы a >= 0: [1 2 0 3 1]
Элементы a <= 0: [-1  0 -2]
Элементы a == 0: [0]
```

Для создания более сложных логических условий можно использовать специальные функции `np.logical_and`, `logical_or`, `logical_not` и `logical_xor`. Они служат заменителями стандартных операторов `and`, `or`, `not` и `xor`. Гораздо удобнее применять побитовые логические операции `&` и `|`. В случае логических масок побитовые операторы полностью заменяют обычные логические операции:

```
# все элементы из отрезка [0, 1]
a[(a >= 0) & (a <= 1)]
```

```
array([1, 0, 1])
```

Комбинируя логические операции с универсальными функциями, можно отфильтровать данные и сразу же применить к ним желаемые операции:

```
a[a>0].sum() # 7
a[(a >= 0) & (a <= 1)].sum() # 2
```

Функция `np.count_nonzero` позволяет вычислить количество ненулевых элементов в массиве. Так как логическое значение `True` интерпретируется как `1`, а значение `False` как `0`, то с помощью `np.count_nonzero` можно проверять массивы на наличие элементов, удовлетворяющих самым разнообразным условиям:

```
# Есть ли элементы из отрезка [0, 1]
np.count_nonzero((a >= 0) & (a <= 1)) # 3
```

Функция `np.any` принимает в качестве аргумента массив с логическими элементами и возвращает истину в случае, если в массиве есть хотя бы один истинный элемент. В свою очередь функция `np.all` возвращает истину, если в массиве все элементы истинны. Эти функции могут заменить `count_nonzero` в случае, если число элементов, удовлетворяющих условию, нам неинтересно:

```
np.any(a > 0), np.all(a > 1)
```

Следует отметить, что в языке `Python` есть встроенные функции `any` и `all`, которые для неочень больших одномерных массивов работают даже быстрее, чем функции `NumPy`. Проверим это утверждение на примере:

```
for i in range(3,7):
    print("Для массива из {0} элементов:".format(i))
    r = np.random.random(10**i)
    %timeit np.any(r>0.5)
    %timeit any(r>0.5)
    %timeit np.all(r>0.5)
    %timeit all(r>0.5)
# Запустите данный код самостоятельно
# в качестве упражнения
```

Однако если `NumPy`-функции можно применять и к многомерным массивам, то встроенные — нельзя, поэтому `NumPy`-функции являются более универсальными.

4.2.2. Чтение и запись данных

Библиотека `NumPy` предоставляет ряд средств, которые позволяют считывать данные из файлов разного формата, а также сохранять массивы в виде файлов для дальнейшего использования. Полный список функций, реализующих считывание и запись файлов, можно найти в разделе Input and output официальной документации по ссылке `https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.io.html`. Мы рассмотрим наиболее часто применяемые из этих функций.

Для преобразования данных, хранящихся в текстовом виде, можно использовать две функции `np.loadtxt` и `np.genfromtxt`. Последняя функция может обработать данные, содержащие ошибки или пропуски. В остальном же эти функции похожи.

Рассмотрим примеры их работы. Создадим вначале текстовый файл `data.csv`, содержащий числовые данные следующего вида:

```
t,x,y
0.1,0.1,0.2
0.2,0.15,0.1
0.3,0.2,0.15
0.4,0.25,0.2
```

Файлы такого формата получили название CSV-файлов (comma separated values — значения, разделённые запятыми) и широко применяются для хранения табличных данных, не требующих форматирования. В качестве разделителя также часто используется точка с запятой. В таком формате удобно хранить результаты не очень объемных вычислений.

Рассмотрим, как с помощью `np.loadtxt` можно извлечь данные из CSV-файла:

```
data = np.loadtxt(fname='data.csv', skiprows=1, delimiter=',')
print(data)
[[ 0.1   1.1   3.2 ]
 [ 0.2   1.15  2.1 ]
 [ 0.3   1.2   1.15]
 [ 0.4   1.25  2.2 ]]
```

Параметр skiprows, равный 1, указывает на то, что при считывании необходимо пропустить первую строку, так как в ней содержатся нечисловые данные (названия колонок). Параметр delimiter задает разделитель. В нашем случае это запятая, но можно задать любой символ. По умолчанию в качестве разделителя используется пробел. По завершении считывания файла функция возвращает двумерный массив. Однако можно считать каждую колонку в отдельную переменную, воспользовавшись параметром unpack:

```
t, x, y = np.loadtxt(fname='data.csv', unpack=True, skiprows=1,
 ↪  delimiter=',')
print(t, x, y)
[ 0.1  0.2  0.3  0.4] [ 1.1   1.15  1.2   1.25] [ 3.2   2.1
 ↪  1.15  2.2 ]
```

Есть возможность указать тип считываемых значений явным образом с помощью аргумента dtype. Причем для разных колонок можно указать разный тип данных. Делается это следующим образом:

```
types = {'names': ('t', 'x', 'y'), 'formats': ('f4', 'i4', 'i4')}
data = np.loadtxt(fname='data.csv', dtype=types, skiprows=1,
 ↪  delimiter=',')
```

В этом случае создается массив специального смешанного типа, к однотипным элементам которого можно обращаться по именам полей, которые мы указали в переменной types. В случае однородных данных достаточно передать dtype один аргумент:

```
data = np.loadtxt(fname='data.csv', dtype=np.float64,
 ↪  skiprows=1, delimiter=',')
```

В случае, если необходимо работать с объемными таблицами с разнородными данными, предпочтительнее использовать библиотеку Pandas, которая специально для этого предназначена.

4.3. Задание

1. С помощью библиотеки NumPy [2] и Jupyter Notebook [3] выполните действия выполнению операций с массивами (см. раздел 4.4).

4.4. Последовательность выполнения работы

Данную лабораторную работу рекомендуется выполнять с помощью Jupyter Notebook, для того чтобы иметь возможность использовать волшебную функцию для замера времени %timeit.

1. Пусть даны два достаточно больших массива, например
   ```
   x = np.random.random(size=10**6)
   y = np.random.random(size=10**6)
   ```
   Найдите поэлементную сумму, разность, произведение и частное для этих двух массивов двумя способами: с помощью цикла и с помощью операторов +, *, −, /. Замерьте и сравните время выполнения этих операций с помощью %timeit, как это сделано в пункте 3.2.2.3.
2. Использует ли функция np.dot несколько ядер процессора для вашей конфигурации NumPy? Для этого запустите выполнение скалярного умножения np.dot(x, y) и проверьте уровень загрузки ядер процессора с помощью команды htop.
3. Найдите максимальный и минимальный элемент массива x с помощью функций np.min и np.max. Сравните быстродействие этих функций со встроенными функциями min и max.

4. Вычислите эмпирические математическое ожидание и дисперсию для массивов `x` и `y`. Чему они равны?
5. Есть ли в массиве `x` элементы меньше 0? Больше 1? Больше 1, 5? Вычислите количество таких элементов.
6. Найдите все элементы из отрезка [0, 1/2] массива `y`.
7. Распечатайте массивы `x` и `y` в формате CSV, где первый столбец содержит элементы `x`, а второй столбец — элементы `y`.

В качестве справочной литературы рекомендуется использовать следующие источники [1; 4; 5].

4.5. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:
- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку цели работы;
- описание результатов выполнения задания:
  - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
  - листинги (исходный код) программ (если они есть);
  - результаты выполнения программ (текст или снимок экрана в зависимости от задания);
- выводы, согласованные с целью работы.

4.6. Контрольные вопросы

Под термином *массив* в вопросах всегда подразумевается массив библиотеки `NumPy`.

1. На каком языке написаны большинство функций `NumPy`?
2. Что такое означает термин «векторизация» в контексте библиотеки `NumPy`?
3. Что такое универсальные функции и как их использование может ускорить программу?
4. Какие две функции позволяют создать линейную последовательность элементов?
5. Как получить доступ к элементам массива с помощью индексов? С какого индекса начинается нумерация? Как получить доступ к последнему индексу?
6. Как работает синтаксис срезов?
7. Как можно эффективно сравнить все соответствующие элементы двух массивов?
8. Как перебрать все элементы массива, пользуясь циклом, но при этом не использовать индексы?
9. Как работают функции `concatenate`, `vstack`, `hstack`?
10. Как работают функции `split`, `hsplit`?
11. Какие математические функции определены в `NumPy`? В каких случаях их стоит применять?
12. При каких операциях `NumPy` может использовать несколько ядер процессора?
13. Какие функции позволяют вычислять среднее, сумму, кумулятивную сумму элементов массива? Могут ли они работать с многомерными массивами?
14. Какие статистические функции `NumPy` вы знаете?
15. Как создать логическую маску и как использовать её для доступа к элементам массива?
16. Какие логические операторы следует использовать для составления сложных выражений при сравнении элементов разных массивов сравнения?

17. Какая функция позволяет вычислить количество ненулевых элементов массива? Как с её помощью вычислить количество любых других одинаковых элементов?
18. Чем функции `any` и `all` из библиотеки `NumPy` отличаются от одноимённых встроенных функций?
19. Какие функции позволяют считать данные из файла? Какого формата данные они могут обрабатывать?
20. Охарактеризуйте формат CSV. Для чего его можно использовать?
21. Могут ли в массивах `NumPy` содержаться разнотипные данные?
22. Как сохранить существующий массив на диск? Какой формат данных при этом можно использовать?
23. Попробуйте сформулировать принципы, следуя которым можно добиться наиболее оптимального в смысле быстродействия, использования библиотеки `NumPy`.
24. Какие функции библиотеки `NumPy` могут пригодится при построении кривых и поверхностей?
25. Пользуясь дополнительной литературой и официальной документацией, ответьте на следующие вопросы:
    - Что такое трансляция (broadcasting) массивов? Какие правила автоматической трансляции использует `NumPy`?
    - Какие функции из библиотеки `NumPy` могут использоваться для сортировки массива?
    - Для чего нужна функция `vectorize`?

Список литературы

1. *Любанович Б.* Простой Python. Современный стиль программирования. — М. : Питер, 2017. — ISBN 978-5-496-02088-6.
2. NumPy home site. — 2018. — URL: `http://www.numpy.org//`.
3. Project Jupyter home. — 2018. — URL: `https://jupyter.org`.
4. Python home site. — 2018. — URL: `https://www.python.org/`.
5. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 5. Основные средства библиотеки Matplotlib для создания изображений и построения графиков

5.1. Цель лабораторной работы

Целью данной лабораторной работы является освоение основных средств создания изображений и построения графиков библиотеки `Matplotlib` [3].

5.2. Предварительные сведения

5.2.1. Библиотека `Matplotlib`

В данном разделе описывается библиотека `Matplotlib`. Изложение ведётся через примеры и пояснения к ним. Естественно, что в ограниченном объёме учебного пособия невозможно описать все аспекты данной библиотеки, поэтому читателю следует в первую очередь обратиться к официальной документации по ссылке `http://matplotlib.org/contents.html`. Кроме того, большую помощь может оказать галерея примеров, также доступная на официальном сайте. Также данной библиотеке посвящены отдельные главы в книгах [1; 5].

5.2.1.1. Общая характеристика библиотеки `Matplotlib`

Библиотека `Matplotlib` является одной из наиболее развитых библиотек для визуализации данных не только в рамках языка `Python`, но и среди других языков программирования и математических пакетов. Сопоставимым уровнем функционала обладают средства визуализации, встроенные в коммерческие программы (MatLab, Mathematica, Maple), библиотека `ggplot` языка `R` и утилита `gnuplot`. Однако `Matplotlib` отличается рядом существенных преимуществ, таких как независимость от платформы, интеграция с библиотекой `NumPy`, поддержка формул LaTeX. Существенной слабой стороной `Matplotlib` является плохая поддержка построения трёхмерных изображений. В остальном данная библиотека является одним из лучших бесплатных решений для визуализации данных.

5.2.1.2. Настройка среды Jupyter

Для использования библиотеки `Matplotlib` совместно с оболочкой `IPython` или `Jupyter` необходимо выполнить команду `%matplotlib inline`, а затем импортировать модуль `matplotlib.pyplot`, дав ему краткое имя `plt`. Именно данный модуль обеспечивает доступ к основным объектам и функциям библиотеки. Также почти во всех примерах будут использоваться функции из библиотеки `NumPy`, так что следует импортировать и эту библиотеку тоже:

```
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib as np
```

Начиная с версии 1.5 в `Matplotlib` появилась возможность использовать отдельные файлы с настройками стилей изображений. Эти файлы должны иметь расширение `.mplstyle` и содержать параметры, аналогичные параметрам из общего файла настроек `.matplotlibrc`. С помощью функции `plt.style.use` можно загрузить и активировать нужный файл стилей. Все рисунки для данного пособия были выполнены с использованием следующего стилевого файла:

```
# black_white.mplstyle
# Настройка стиля для черно-белых картинок
figure.figsize: 10, 4.0
figure.dpi: 200
# Настройка шрифтов
font.family: serif
font.serif: DejaVu Serif, Times New Roman
font.sans-serif: DejaVu Sans, Arial
font.monospace: DejaVu Sans Mono, Consolas, Courier New
font.size: 11
lines.markersize: 3
# Настройки сетки координат
axes.grid: True
grid.linewidth: 0.5
grid.linestyle: dashed
grid.color: gray
axes.prop_cycle: ((cycler('color', ['k']) * cycler('linestyle',
 ↪  ['solid', 'dashed', 'dashdot', 'dotted'])) *
 ↪  cycler('marker', [' ', '.', '^']))
```

5.2.2. Создание изображения и системы координат

Изначально `Matplotlib` разрабатывался с оглядкой на `MatLab`, поэтому в библиотеку встроен интерфейс, во многом повторяющий `MatLab`. Мы, однако, будем пользоваться объектно-ориентированным интерфейсом `Matplotlib`, так как он обеспечивает большую гибкость и контроль над настройками графиков.

Построение любого графика начинается с создания изображения (объект класса `figure`) и добавления на это изображение одной или нескольких систем координат (объекты класса `axes`). Для создания изображения используется метод `plt.figure()`, который возвращает объект, представляющий собой пустое изображение без осей, графиков и надписей. В дальнейшем можно добавлять на это изображение необходимые элементы. Метод `figure` может принимать ряд аргументов, полный список которых можно посмотреть в справочной строке данного метода (в Jupyter Notebook нажать клавишу Tab после имени метода).

Следующим шагом необходимо добавить в объект `figure` оси координат. Стандартный метод `add_subplot()` позволяет задать прямоугольную сетку из субкоординат (рис. 5.1):

```
fig01 = plt.figure(num=0)
ax01 = fig01.add_subplot(2, 1, 1)
ax02 = fig01.add_subplot(2, 1, 2)
```

В данном примере изображение (см. рис. 5.1) разбивается на две части по горизонтали. Первый аргумент `add_subplot()` — число строк, второй число столбцов и третий — номер субграфика. С каждым субграфиком можно взаимодействовать отдельно (переменные `ax01` и `ax02`), строя на нем кривые, меняя тип координатных линий, добавляя аннотации, легенды и подписи к осям.

Добавим к созданным системам координат графики синуса и косинуса (рис. 5.2). Для этого сначала рассчитаем соответствующие значения:

```
t = np.linspace(-4*np.pi, 4*np.pi, 400)
x = np.sin(t)
y = np.cos(t)

fig01 = plt.figure(num=0)
```

```
ax01 = fig01.add_subplot(2, 1, 1)
ax02 = fig01.add_subplot(2, 1, 2)

ax01.plot(t, x)
ax02.plot(t, y)
```

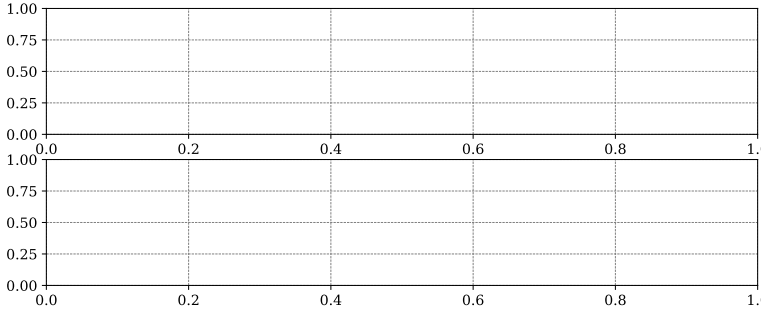

Рис. 5.1. Два субграфика с прямоугольной сеткой

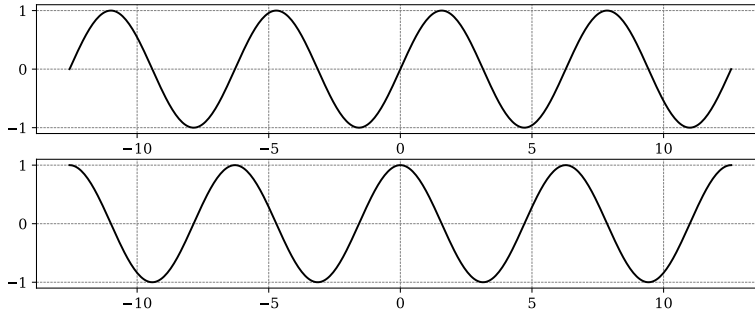

Рис. 5.2. Графики синуса и косинуса на двух субграфиках

Если необходимо создать большое число субграфиков в сетке, как показано на рис. 5.3, то удобно сделать это в цикле или в списковой сборке:

```
fig02 = plt.figure(num=1)
ax = [fig02.add_subplot(2, 2, i) for i in range(1,5)]

# аналогичный способ с циклом
# ax = []
# for i in range(1,5):
#     ax.append(fig02.add_subplot(2, 2, i))
```

В следующем примере (рис. 5.4) на каждом субграфике строится функция $y = \sin(ix) + \cos(ix)$, где i — номер субграфика (начиная с 0):

```
fig03 = plt.figure(num=2)
axs = [fig03.add_subplot(3, 3, i) for i in range(1,10)]

for (i, ax) in enumerate(axs):
    y = np.sin(i*x) + np.cos(i*x)
    ax.plot(x, y, linewidth=0.4)

fig03.tight_layout()
```

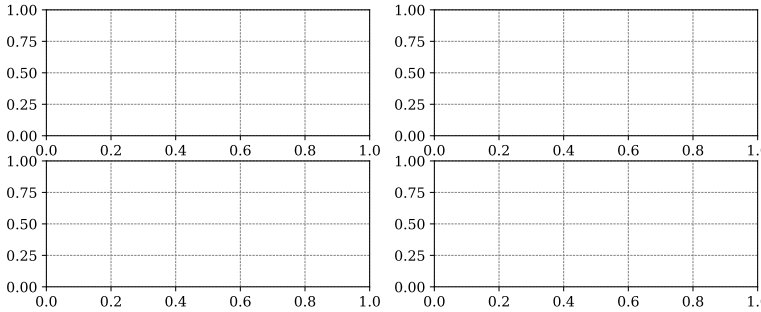

Рис. 5.3. Четыре субграфика с прямоугольной сеткой

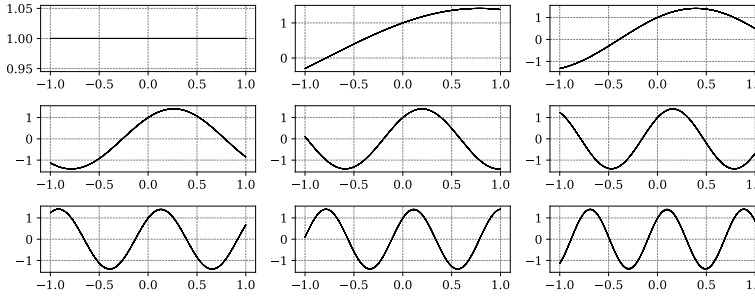

Рис. 5.4. Графики функции $y = \sin(ix) + \cos(ix)$, где i — номер субграфика

5.2.3. Метод `plot`

Метод `plot` служит основным средством для создания одномерных графиков из массива координатных точек. Кроме непосредственно набора координат он может принимать множество параметров для тонкой настройки вида графика. Перечислим наиболее часто используемые из них:

- `color` — цвет; можно задавать любой цвет, если строить несколько графиков, то они автоматически будут разного цвета;
- `linewidth` — толщина линии, по умолчанию равна 1;

– `linestyle` — тип линии кривой; среди поддерживаемых типов сплошная линия (`solid` или `-`), пунктирная (`dashed` или `--`), точечная (`doted` или `:`) и пунктирно-точечная (`dashdot` или `-.`);
– `label` — легенда графика (то есть краткое пояснение к графику, обычно отображающееся в углу изображения); для отображения легенды следует вызвать метод `legend` объекта `axes`.

Рассмотрим пример использования метода `plot` для отображения двух графиков в одной системе координат в черно-белом формате (рис. 5.5):

```
t = np.linspace(-4*np.pi, 4*np.pi, 400)
x = np.sin(t) + np.cos(t) + np.cos(2*t)
y = np.cos(2*t) + np.sin(np.cos(t))

fig04 = plt.figure(10)
ax04 = fig04.add_subplot(1, 1, 1)
label_01 = r'$\sin({t}) + \cos({t}) + \cos({2t})$'
label_02 = r'$\cos{(2t)} + \sin{[\cos{(t)}]}$'
ax04.plot(t, y, linewidth=1.0, linestyle='-', marker='None',
 ↪  label=label_01)
ax04.plot(t, x, linewidth=0.5, linestyle='-', marker='.',
 ↪  markevery=5, markersize=4,  label=label_02)

ax04.legend(ncol=2, title='Легенда', framealpha=0.7, fontsize=12)
```

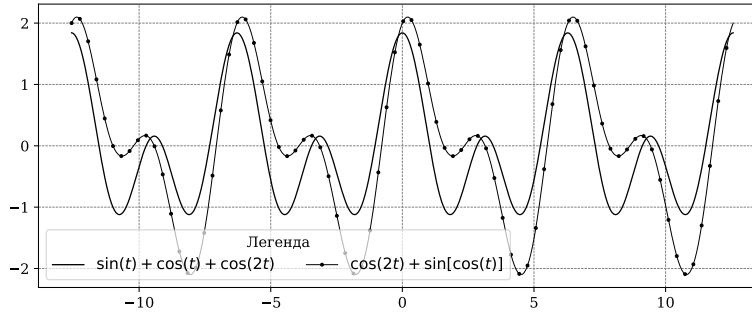

Рис. 5.5. Два графика в одной системе координат

В данном примере кроме вышеперечисленных опций функции `plot` были использованы опции для настройки маркеров — значков, которыми можно выделить точки графиков. Параметр `marker` устанавливает значок маркера (полный список смотрите в документации к `plot`), параметр `markevery` позволяет регулировать количество маркеров, отображаемых на кривой, `markersize` позволяет установить размер маркера.

В методе `legend` мы использовали параметр `ncol`, который устанавливает количество колонок в легенде, параметр `title` устанавливает заголовок для легенды (по умолчанию заголовок не отображается), параметр `framealpha` регулирует прозрачность рамки легенды, `fontsize` устанавливает размер шрифта. Последним параметром не стоит злоупотреблять, так как гораздо удобнее установить один размер шрифта для всех элементов графика (как это сделать, мы рассмотрим далее).

5.2.4. Настройка осей координат

Продолжим настраивать наш график из предыдущего примера. Добавим подписи к осям Ox и Oy с помощью методов `set_xlabel` и `set_ylabel`, заголовок к изображению с помощью метода `set_title`, а также настроим точные границы осей координат с помощью метода `set_xlim`:

```
ax04.set_title('Это заголовок нашей картинки')
ax04.set_xlabel(r'Ось $x$')
ax04.set_ylabel(r'Ось $y$')
ax04.set_xlim(left=t[0], right=t[-1])
```

Проведем теперь более сложные настройки и заменим отсечки координат по оси `Ox` не просто вещественными числами, а числами, кратными π. Для этого сначала создадим два списка. В `xticks` запишем точки -4π, -3π, -2π, $-\pi$, 0, π, 2π, 3π, 4π, а в список `xtickslabels` запишем значения строкового типа, которые предполагается отображать в качестве обозначения отсечек. Можно использовать команды LaTeX для отображения буквы π или юникодный символ π. После того как списки созданы, их надо передать методам `set_xticks` и `set_xticklabels`, которые заменят автоматически созданные отсечки на созданные нами:

```
xticks = np.linspace(-4*np.pi, 4*np.pi, 9)
xtickslabels = []
for xtick in xticks:
    i = int(xtick/np.pi)
    if i == 1:
        xtickslabels.append(r'$\pi$')
    elif i == -1:
        xtickslabels.append(r'-$\pi$')
    elif i == 0:
        xtickslabels.append(r'0')
    else:
        xtickslabels.append(r'{0}$\pi$'.format(i))
ax04.set_xticks(xticks)
ax04.set_xticklabels(xtickslabels)
fig04
```

Результат показан на рис. 5.6.

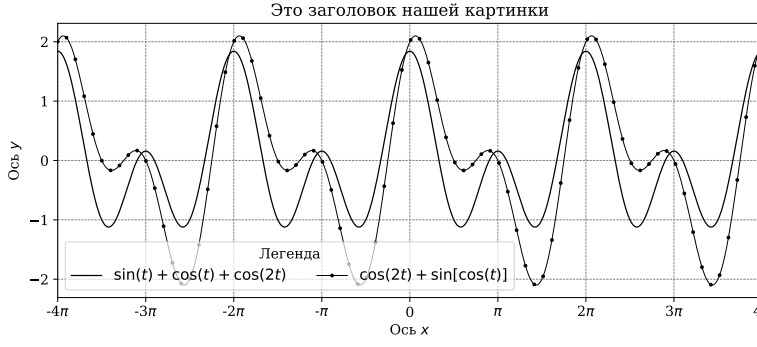

Рис. 5.6. Графики рис. 5.5 после модификации осей

Часто бывает необходимо построить несколько графиков в отдельных системах координат, но при этом близко друг к другу с общими отсечками по одной из осей (рис. 5.7):

```
fig05 = plt.figure(num=4, figsize=(10, 2))

axs05 = [fig05.add_subplot(1, 3, i) for i in range(1,4)]

x = np.linspace(-2,2,100)
funcs = [x, x**2, x**3]

y_min = np.min(funcs)
y_max = np.max(funcs)

for (f, ax) in zip(funcs, axs05):
    ax.plot(x, f)
    ax.set_ylim(bottom=y_min, top=y_max)

for ax in axs05[1:]:
    ax.set_yticklabels([])

fig05.subplots_adjust(wspace=0)
```

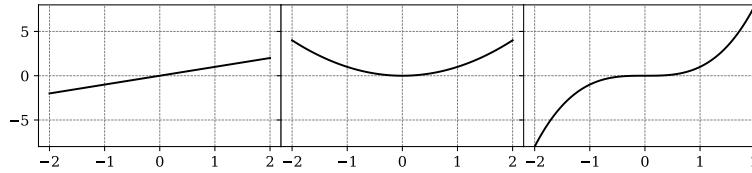

Рис. 5.7. Несколько графиков в отдельных системах координат с общими отсечками по одной из осей

5.2.5. Дополнительные способы создания сетки субграфиков

Если необходимо создать большое количество субграфиков, то можно воспользоваться функцией `plt.subplots`. Эта функция создаст целую сетку субграфиков и вернет объект `figure` и массив объектов `axes`. В качестве аргументов `subplots` принимает количество строк и столбцов, а также необязательные логические аргументы `sharex` и `sharey`, которые позволяют установить общую разметку координат. Проиллюстрируем работу функции на примере (рис. 5.8):

```
rows_number = 3; columns_number = 4
fig06, ax06  = plt.subplots(nrows = rows_number, ncols =
 ↪  columns_number, sharex=True, num=6)

for (i, j), ax in np.ndenumerate(ax06):
    ax.set_title("(стр.: {0}, кол.: {1}).".format(i+1, j+1))

t = np.linspace(-2*np.pi, 2*np.pi, 200)
ticklabels = ['$-2\pi$','$-\pi$','$0$','$\pi$','$2\pi$']
```

```
for (i, j), ax in np.ndenumerate(ax06):
    ax.plot(t, np.sin((i+j+1)*t), linewidth=0.5)
    ax.xaxis.set_ticks(np.linspace(-2*np.pi,2*np.pi,5))
    ax.xaxis.set_ticklabels(ticklabels)

fig06.tight_layout()
```

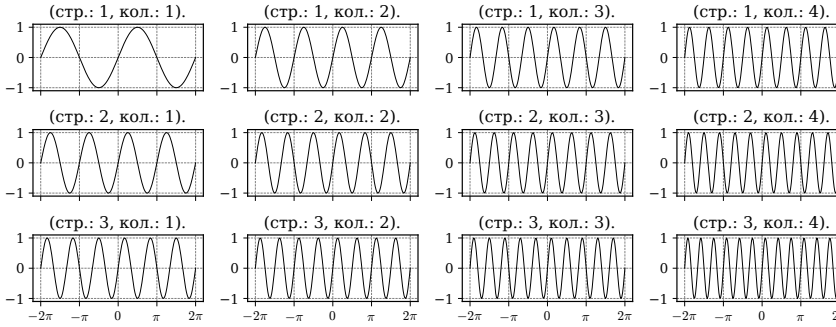

Рис. 5.8. Построение графиков с применением функции `subplots`

Для обхода массива `ax06` мы использовали функцию `ndenumerate` библиотеки `NumPy`, которая работает аналогично встроенной функции `enumerate`, но возвращает в случае двумерного массива кортеж индексов `(i, j)`, который мы сразу же распаковываем в соответствующие индексы. Опция `sharex=True` убирает разметку оси *Ox* для всех субграфиков кроме нижнего ряда. Это экономит место и делает изображение более чистым.

Если необходимо задать неоднородную сетку субграфиков, то можно воспользоваться функцией `gridspec.GridSpec`. Для этого необходимо предварительно импортировать соответствующий подмодуль **`import matplotlib.gridspec as gridspec`**. Эта функция возвращает массив, с помощью которого можно затем задать местоположение и размеры субграфиков, используя обычный синтаксис срезов языка Python. Применение `GridSpec` иллюстрирует следующий пример (рис. 5.9):

```
fig07 = plt.figure(num=7)

gs = gridspec.GridSpec(3, 3, left=0.0, right=0.9, wspace=0.4,
 ↪  hspace = 0.7)

ax07 = []
ax07.append(fig07.add_subplot(gs[0, 0:]))
ax07.append(fig07.add_subplot(gs[1, 0:1]))
ax07.append(fig07.add_subplot(gs[1, 1:]))
ax07.append(fig07.add_subplot(gs[2, 0:2]))
ax07.append(fig07.add_subplot(gs[2, 2:]))

for i, ax in enumerate(ax07):
    ax.set_title("Заголовок {0}".format(i))
    ax.set_ylabel("$Oy$")
```

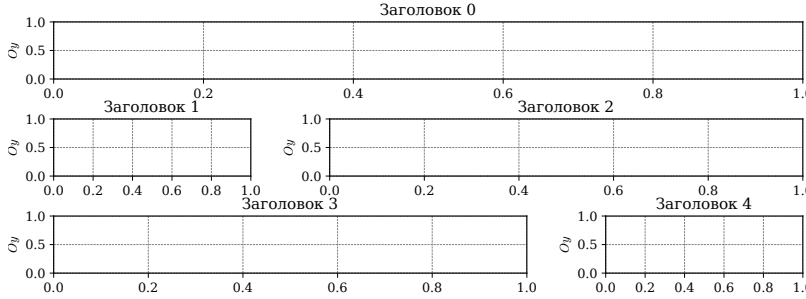

Рис. 5.9. Построение графиков с применением `GridSpec`

Наконец, максимально гибкое средство настройки сетки субграфиков произвольного вида — это функция `plt.axes`. Данная функция принимает необязательный аргумент, представляющий собой список из четырех чисел в системе координат рисунка. Эти числа означают левый угол, низ, ширину и высоту в системе координат рисунка, отсчет которых начинается с 0 в нижнем левом и заканчивается 1 в верхнем правом углу рисунка. Первая пара из списка фактически представляет собой координаты левого нижнего угла прямоугольника, а вторая — координаты правого верхнего угла.

С помощью метода `axes` можно встраивать малые субграфики прямо в область координат большого графика, что может пригодится для отображения увеличенной окрестности особых точек, например, точек пересечения двух кривых, как в примере ниже (рис. 5.10):

```
fig08 = plt.figure(num=8)

ax08 = [
    fig08.add_axes([0.0, 0.0, 1.0, 1.0]),
    fig08.add_axes([0.20, 0.6, 0.2, 0.3], xlim=(-1.5, 0),
     ↪  ylim=(-3, 10)),
    fig08.add_axes([0.65, 0.6, 0.2, 0.3], xlim=(1.5, 2.1),
     ↪  ylim=(0, 10))
]

t = np.linspace(-2.0, 2.1, 500)
y = t**4 - t**3 + t**2 - t - 1
z = t**3 - t**2 + 3

for ax in ax08:
    ax.plot(t, y, linewidth=1.0)
    ax.plot(t, z, linewidth=1.0, markevery=10)
    ax.set_xlabel(r'Ось $x$'); ax.set_ylabel(r'Ось $y$')
```

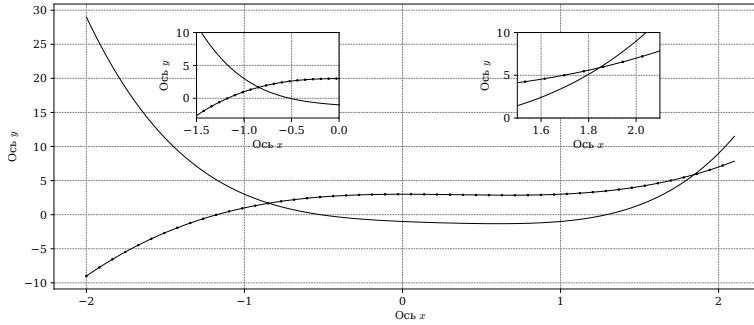

Рис. 5.10. Построение графиков с применением `add_axes`

5.2.6. Манипуляция осями координат. Полярные координаты

Как видно из предыдущих примеров, по умолчанию координатные оси отображаются не по центру, а сбоку. Чаще всего такое расположение осей является оптимальным, так как не мешает восприятию графической информации. Однако есть возможность переместить оси координат в центр графика. Как это сделать, показывает следующий пример (рис. 5.11):

```
fig09 = plt.figure(num=9)
ax09 = fig09.add_subplot(1, 1, 1)

t = np.linspace(-4*np.pi, 4*np.pi, 200)
y = np.sin(t)
ax09.plot(t, y)
# Убираем верхнюю и правую оси координат
ax09.spines['top'].set_visible(False)
ax09.spines['right'].set_visible(False)

# 'center' -> ('axes',0.5) 'zero' -> ('data', 0.0)
# Переносим оставшиеся две оси в центр относительно графика
ax09.spines['bottom'].set_position('zero')
ax09.spines['left'].set_position('zero')

# настраиваем засечки на координатных линиях
ax09.xaxis.set_tick_params(direction='inout', length=10)
ax09.yaxis.set_tick_params(direction='inout', length=10)
```

В Matplotlib есть возможность использовать полярную систему вместо декартовой. Рассмотрим на примере, как это сделать (рис. 5.12):

```
fig10 = plt.figure(num=10, figsize=(3,3))
ax10 = fig10.add_subplot(1, 1, 1, projection='polar')
phi = np.linspace(0.0, 10*np.pi, 1000)
r = 5*np.cos(2.5*phi-1)
ax10.plot(phi, r, linewidth=1.0)
```

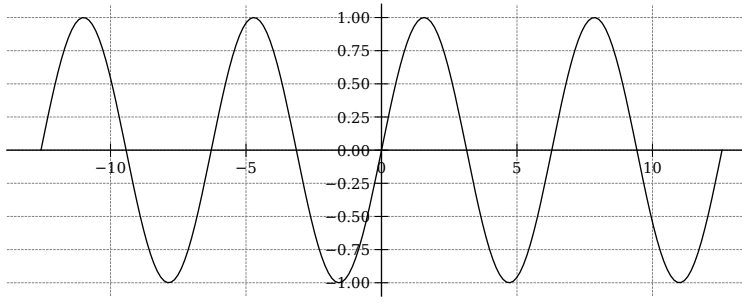

Рис. 5.11. Расположение осей координат по центру

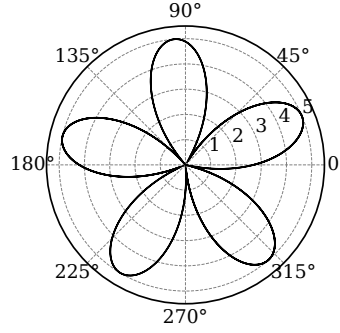

Рис. 5.12. Построение графика в полярной системе координат

5.3. Задание

1. Настройте используемое вами окружение для работы с библиотекой `Matplotlib`.
2. С помощью библиотеки `Matplotlib` постройте систему координат сначала с использованием интерактивной оболочки, затем без её использования (см. раздел 5.4.1).
3. Выполните задания по построению субграфиков из раздела 5.4.2.

5.4. Порядок выполнения лабораторной работы

5.4.1. Построение системы координат

1. В случае использования интерактивной оболочки импортируйте модуль `Matplotlib`, предварительно выполнив «волшебную» команду `%matplotlib inline`.
2. Для построения системы координат выполните следующий код:

```
% matplotlib inline
import matplotlib.pyplot as plt
```

```
fig = plt.figure(num=0)
ax = fig.add_subplot(1, 1, 1)
```

3. Создайте файл с расширением `.py`, запишите в него код, приведённый в предыдущем пункте, добавив сохранение изображения в виде файла:
```
fig.savefig('img.png', dpi=300, format='png')
```
и запустите на выполнение.

5.4.2. Построение субграфиков

В процессе выполнения последующих заданий необходимо использовать код, приведённый в примерах раздела 5.2. Рекомендуется перед внесением изменений запустить исходный код примеров и убедиться, что получившиеся изображения совпадают с теми, которые приведены в данном разделе, и только после этого приступать к модификации кода.

1. Используйте пример построения изображения `fig01` (см. рис. 5.1 и 5.2). Измените его так, чтобы изображение разделилось на две координатные системы не горизонтально, а вертикально.
2. Используйте пример построения изображения `fig04` (см. рис. 5.5 и 5.6). Измените цвет построенных графиков и уберите маркеры.
3. Используйте пример построения изображения `fig05` (см. рис. 5.7). Замените горизонтальное расположение субграфиков на вертикальное.
4. Взяв способ построения субграфиков из примера для изображения `fig08` (см. рис. 5.10), с его помощью воссоздайте рис. 5.10 без заголовков осей координат.
5. В примере `fig09` (см. рис. 5.11) передвиньте ось Oy в крайне правое положение, а ось Ox верните в нижнюю часть изображения.
6. В примере `fig10` (см. рис. 5.12) замените нарисованную кривую на спираль Архимеда $r = \frac{1}{2}\varphi$.

В качестве справочной литературы рекомендуется использовать следующие источники [2; 4; 5].

5.4.3. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:

- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку цели работы;
- описание результатов выполнения задания:
  - полученные вами графические изображения;
  - листинги изменённых вами примеров с указанием того, какие части примеров были вами изменены и с какой целью это было сделано;
- выводы, согласованные с целью работы.

5.5. Контрольные вопросы

1. Для чего предназначена библиотека `Matplotlib`?
2. Какие преимущества есть у библиотеки `Matplotlib` по сравнению с конкурентами?
3. Какая команда позволяет встраивать графики, построенные с помощью `Matplotlib`, прямо в интерактивный блокнот `Jupyter`?

4. Как можно настроить внешний вид графиков, используемый `Matplotlib` по умолчанию?
5. Сколько осей координат может быть на одном изображении?
6. Что такое субграфик в терминах `Matplotlib`?
7. Какая функция является основной для построения плоских кривых?
8. Какие методы позволяют создавать сетки субкоординат (субграфиков)?
9. Какие параметры можно передавать функции `plot` для настройки внешнего вида строящихся кривых? Перечислите некоторые из них.
10. Как в `Jupyter` получить доступ к строке документации той или иной функции?
11. Какой метод следует вызвать, чтобы отобразить легенду для построенных графиков?
12. Какие настройки есть у легенды? Как добавить заголовок легенды? Как изменить её расположение на рисунке?
13. Какие методы позволяют установить подписи к осям и заголовок?
14. Каким образом в текстовые элементы, располагаемые на осях, можно добавить формулы в формате LaTeX?
15. Какие настройки регулируют цвет, размер и форму маркеров? Как настроить частоту расстановки маркеров на кривой?
16. Какие методы регулируют отображение символов разметки осей координат?
17. Какие методы позволяют манипулировать осями координат и регулировать их видимость?
18. Как построить график в полярной системе координат?

Список литературы

1. *Маккинни У.* Python и анализ данных. — М. : ДМК Пресс, 2015. — ISBN 978-5-97060-315-4.
2. *Любанович Б.* Простой Python. Современный стиль программирования. — М. : Питер, 2017. — ISBN 978-5-496-02088-6.
3. Project Jupyter home. — 2018. — URL: `https://jupyter.org`.
4. Python home site. — 2018. — URL: `https://www.python.org/`.
5. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 6. Расширенные средства библиотеки Matplotlib для создания изображений и построения графиков

6.1. Цель лабораторной работы

Целью данной лабораторной работы является более глубокое знакомство со средствами создания изображения библиотеки `Matplotlib` [3].

6.2. Предварительные сведения

6.2.1. Создание примитивов

Под термином примитив понимаются простейшие геометрические объекты: отрезки, окружности, треугольники и прямоугольники. Для их рисования можно использовать специальные методы, которые позволят, например, добавить на рисунок окружность, указав лишь координаты ее центра и радиус. В `Matplotlib` функции для создания примитивов находятся в подмодуле `patches` и `lines`.

Перечислим некоторые из функций подмодуля `patches`:

- `Circle` — создает окружность с центром в точке с координатами `xy` и радиусом `radius`;
- `Ellipse` — создает эллипс с центром с координатами `xy` и полуосями `width` и `height`;
- `Rectangle` — создает прямоугольник, левый нижний край которого имеет координаты `xy`; высота и ширина прямоугольника регулируются параметрами `width` и `height`, а также его можно вращать вокруг левого нижнего угла, задав параметр `angle` (в градусах);
- `Arc` — создает дугу эллипса; кроме параметров самого эллипса (см. `Ellipse`) принимает параметры `theta1` и `theta2`, которые задают начальный и конечный углы дуги, а также параметр `angle`, который позволяет вращать дугу вокруг центра как единое целое;
- `Wedge` — создает клин (сегмент окружности) с центром в точке `center` и радиусом `r`; отличие от дуги эллипса `Arc` заключается в прорисовке радиусов; размер центрального угла задается параметрами `theta1` и `theta2`, так же как у дуги эллипса;
- `Polygon` — ломаная линия (замкнутая или не замкнутая) произвольного вида, задаваемая массивом координат `xy`; принимает логический параметр `closed`, который отключает или включает автоматическое замыкание ломаной (соединяет начальную и последнюю точки);
- `Arrow` и `FancyArrow` — создают стрелку с началом в точке с координатами `x`, `y` и проекциями длины `dx` по оси Ox и `dy` по оси Oy; `FancyArrow` дополнительно позволяет более гибко настроить внешний вид стрелки;
- `FancyBboxPatch` — создает прямоугольник со скругленными краями.

Каждая из вышеперечисленных функций принимает в качестве необязательных параметров стандартные аргументы, управляющие внешним видом линии: `linestyle`, `linewidth`, `color` и т.д. Особенно полезен логический аргумент `fill`, который включает или отключает заливку фигуры сплошным цветом.

После создания примитива его необходимо добавить на изображения. Для этого используется метод `add_patch` объекта `axes`. Пример ниже показывает использование всех вышеперечисленных примитивов (рис. 6.1).

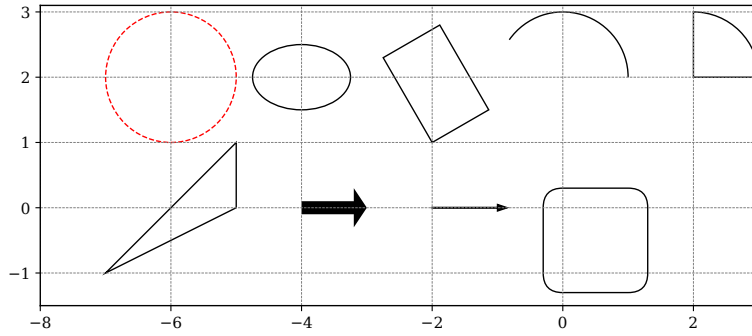

Рис. 6.1. Доступные в `Matplotlib` примитивы

```
fig11 = plt.figure(num=11)
ax11 = fig11.add_subplot(1, 1, 1)

ax11.set_xlim(left=-8, right=3.1)
ax11.set_ylim(bottom=-1.5, top=3.1)

ax11.set_aspect('equal')
# можно использовать универсальный метод set
# ax11.set(xlim=(-2, 2), ylim=(-2, 2), aspect='equal')

# Окружность
circle = mpatches.Circle(xy=(-6, 2), radius = 1, color='red',
 ↪  linestyle='--', fill=False)
# Дуга эллипса
elipse = mpatches.Ellipse(xy=(-4, 2), width=1.5, height=1,
 ↪  fill=False)
# Прямоугольник
rectangle = mpatches.Rectangle(xy=(-2, 1), width=1, height=1.5,
 ↪  angle=30, fill=False)
# Арка эллипса
arc = mpatches.Arc(xy=(0, 2), width=2, height=2, angle=0,
 ↪  theta1=0, theta2=145)
# Сегмент окружности (клин)
wedge = mpatches.Wedge(center=(2, 2), r=1, theta1=0, theta2=90,
 ↪  fill=False)
# Полигон -- ломаная линия (замкнутая или незамкнутая)
 ↪  произвольного вида.
polygon = mpatches.Polygon(xy=np.array([[-7, -1], [-5, 1], [-5,
 ↪  0]]), closed=True, fill=False)
# Стрелка
arrow = mpatches.Arrow(x=-4, y=0, dx=1, dy=0)
# Стрелка с возможностью настройки внешнего вида
farrow = mpatches.FancyArrow(x=-2, y=0, dx=1, dy=0, width=0.01,
 ↪  head_width=0.1)
# Прямоугольник с закругленными углами
```

```
fbox = mpatches.FancyBboxPatch(xy=(0, -1), width=1, height=1.0,
 ↪  fill=False)

ax11.add_patch(circle); ax11.add_patch(elipse);
 ↪  ax11.add_patch(rectangle); ax11.add_patch(arc)
ax11.add_patch(wedge); ax11.add_patch(polygon);
 ↪  ax11.add_patch(arrow); ax11.add_patch(farrow);
 ↪  ax11.add_patch(fbox)
```

В подмодуле `lines` находятся низкоуровневые функции, применяемые для соединения точек ломаными или гладкими кривыми разного порядка. Из всех этих функций мы рассмотрим лишь одну — `Line2D`. Данная функция принимает два массива координат: по оси *Ox* (`xdata`) и по оси *Oy* (`ydata`). Кроме этих параметров есть возможность очень гибко настроить вид линии. В примере ниже использован параметр `marker`, который добавляет маркеры в каждую из вершин получившейся ломаной (рис. 6.2).

```
fig12 = plt.figure(num=12, figsize=(10, 1.5))
ax12 = fig12.add_subplot(1, 1, 1)

ax12.set_xlim(left=-1.1, right=1.1)
ax12.set_ylim(bottom=-0.1, top=1.1)

dots = {'xdata': [-1, 0, 1], 'ydata': [0, 1, 0]}
line = mlines.Line2D(**dots, marker='o')

ax12.add_line(line)
```

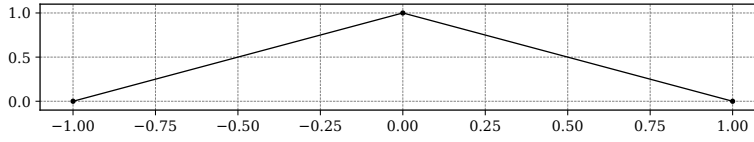

Рис. 6.2. Применение `Line2D`

Обратите внимание, что сначала мы создали словарь `dots`, а потом передали его в функцию с помощью оператора `**`. Данный оператор распаковывает словарь и передает в функцию именованные аргументы с теми же именами, что и ключи словаря. Это полезно в случае, если мы вычисляем точки ломаной отдельно, например, считываем из файла или получаем в `JSON`-формате. Естественно, что можно обойтись более стандартным синтаксисом и записать следующий код:

```
xs = [-1, 0, 1]
ys = [0, 1, 0]
line = mlines.Line2D(xdata=xs, ydata=ys, marker='o')
```

6.2.2. Аннотирование изображений

6.2.2.1. Текстовые пометки

Библиотека `Matplotlib` предоставляет широкие возможности по аннотированию созданных графиков. Основными средствами для этого служат методы `text` и `annotate` класса `axes`.

Метод `text` позволяет добавить текст возле любой точки на координатной плоскости. При этом внешний вид текста можно гибко настраивать, в том числе включать LaTeX-формулы. Метод `text` требует три обязательных аргумента: `x`, `y` — координаты точки, относительно которой будет отображаться текст, и строка `s`, содержащая непосредственно сам текст. Кроме этих аргументов можно передавать большое число дополнительных параметров. Некоторые из них показаны в примере:

```
fig13 = plt.figure(num=13)
ax13 = fig13.add_subplot(1, 1, 1)

ax13.set_xlim(0, 1.5)
ax13.set_ylim(0, 1.5)

points = [(0.25, 0.25), (0.50, 0.50), (0.75, 0.75), (1.0, 1.0),
 ↪  (1.25, 1.25)]

xs = [p for (p, _) in points]
ys = [p for (_, p) in points]

ax13.plot(xs, ys, marker='o', markersize=6, linestyle='None')

x, y = points[0]
ax13.text(x=x, y=y+0.1, s='Точка №1', color='gray')

x, y = points[1]
ax13.text(x=x, y=y+0.1, s='Точка №2', rotation_mode='anchor',
 ↪  rotation=90)

x, y = points[2]
ax13.text(x=x, y=y+0.1, s='Точка №3',
 ↪  horizontalalignment='center', fontstyle='italic',
 ↪  fontsize=14)

x, y = points[3]
ax13.text(x=x, y=y+0.1, s='Точка №4', fontweight='extra bold')

x, y = points[4]
ax13.text(x=x, y=y+0.1, s=r'$\mathscr{{P}}({0}, {1})$'.format(x,
 ↪  y))
ax13.text(x=0.2, y=1.2, s=r'$\lim_{x \to +
 ↪  \infty}\left(\frac{\pi(x)}{x/\ln x}\right)=1$', fontsize=20)
```

В примере и на рис. 6.3 показаны различные возможности по форматированию и преобразованию отображаемого на координатной плоскости текста. Для первой точки изменён цвет текста на серый. Для второй точки надпись повёрнута на 90 градусов вокруг точки (если не использовать `rotation_mode='anchor'`, то надпись будет повёрнута вокруг своего центра). У текста возле третьей точки установлено выравнивание по центру, курсивное начертание и увеличенный размер шрифта. Для четвёртой точки надпись выполнена жирным шрифтом. Наконец, пятая точка обозначена буквой $\mathscr{P}$ со специальным шрифтом с помощью команды LaTeX. Также отдельно на координатной плоскости записано математическое выражение в LaTeX-нотации.

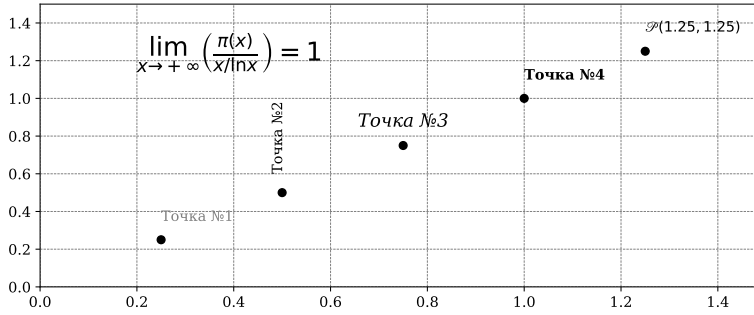

Рис. 6.3. Текстовые средства `Matplotlib` для добавления пояснений на изображение

6.2.2.2. Аннотации

Аннотация представляет собой рамку с текстом и стрелку, которая отходит от рамки по направлению к аннотируемому объекту. Возможности по настройке внешнего вида аннотации практически безграничны, так как имеется возможность настраивать отдельно внешний вид текста, рамки и стрелки. Представление о всех возможностях можно получить на странице документации по ссылке `https://matplotlib.org/users/annotations.html`.

Пример (рис. 6.4):

```
fig14 = plt.figure(num=14)

ax14 = fig14.add_subplot(1,1,1)

ax14.set_xlim(0, 1.5)
ax14.set_ylim(0, 1.5)

points = [(0.25, 0.25), (0.50, 0.50), (0.75, 0.75), (1.0, 1.0),
 ↪  (1.25, 1.25)]

xs = [p for (p, _) in points]
ys = [p for (_, p) in points]

ax14.plot(xs, ys, marker='o', markersize=6, linestyle='None')

x, y = points[0]
ax14.annotate('Точка №1', xy=(x, y), xytext=(-25, 25),
 ↪  textcoords='offset points', arrowprops=dict(arrowstyle='->',
 ↪  lw=2, color='gray'), bbox=dict(boxstyle='round',
 ↪  fc='lightgray'))

x, y = points[1]
# Если не указать xytext, то аннотация будет расположена прямо
 ↪  около точки
ax14.annotate('Точка №2', xy=(x, y), bbox=dict(boxstyle='round',
 ↪  fill=False))
```

```
x, y = points[2]
ax14.annotate('Точка №3', fontname='Comic Sans MS', xy=(x, y),
 ↪  horizontalalignment='center', xytext=(-100, 50),
 ↪  textcoords='offset points', arrowprops=dict(arrowstyle='->',
 ↪  lw=1, connectionstyle='angle3'),
 ↪  bbox=dict(boxstyle='circle', fc='lightgray'))

ax14.annotate('Точка №3', fontname='Consolas', xy=(x, y),
 ↪  horizontalalignment='center', xytext=(100, -95),
 ↪  textcoords='offset points',
 ↪  arrowprops=dict(arrowstyle='<->', lw=1,
 ↪  connectionstyle='angle'),
 ↪  bbox=dict(boxstyle='round4',fill=None))

x, y = points[3]
ax14.annotate('Точка №4', fontname='Mistral', xy=(x, y),
 ↪  fontsize=16, horizontalalignment='center', xytext=(100,
 ↪  -95), textcoords='offset points',
 ↪  arrowprops=dict(arrowstyle='-|>', linestyle='--'),
 ↪  bbox=dict(boxstyle='roundtooth', color='black', fill=None))

x, y = points[4]
ax14.annotate('Точка №4', xy=(x, y), xytext=(0, -70),
 ↪  textcoords='offset points', horizontalalignment='center',
 ↪  arrowprops=dict(arrowstyle='fancy', lw=0.5, fill=None),
 ↪  bbox=dict(boxstyle='darrow', lw=0.5, color='black',
 ↪  fill=None))
```

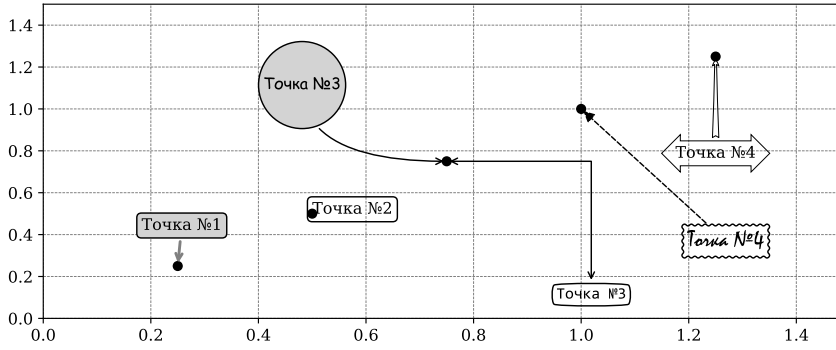

Рис. 6.4. Некоторые из большого числа возможных стилей аннотаций

6.2.3. Дополнительные средства для построения графиков

До сих пор во всех примерах для построения графиков мы пользовались лишь функцией `plot`. Данная функция благодаря гибкой настройке позволяет строить разнообразные графики. Однако библиотека `Matplotlib` не ограничивается лишь

одной этой функцией и предлагает большое разнообразие средств для визуализации, среди которых — функции для построения гистограмм, спектров, круговых и столбчатых диаграмм, графиков с планками погрешностей, ступенчатых графиков, векторных полей, вертикальных и горизонтальных линий, диаграмм рассеяния и т.д. Описать их все здесь не представляется возможным, так что мы ограничимся лишь несколькими.

6.2.3.1. Построение поля векторов

Для построения поля векторов используется специальная функция `quiver` (буквально «колчан стрел», что иносказательно намекает на множество стрелок/векторов). Функцию `quiver` можно использовать для изображения как одного вектора, так и целого набора векторов, например, касательных в каждой точке некоторой кривой. В качестве обязательных аргументов функция принимает два массива координат векторов по оси Ox (аргумент `U`) и по оси Oy (аргумент `V`). Следующие два аргумента — массивы `X` и `Y` — это наборы координат начала каждого из векторов. Если их опустить, то векторы будут распределены по равномерной сетке.

В качестве примера изобразим касательные векторы для графика функции $\sin(x)$ (рис. 6.5), параметрическое уравнение которой можно записать следующим образом:

$$\begin{cases} x = t, \\ y = \sin(t), \\ t \in \mathbb{R}, \end{cases} \quad \begin{cases} x' = 1, \\ y' = \cos(t), \\ t \in \mathbb{R}. \end{cases}$$

```
t = np.linspace(-2*np.pi, 2*np.pi, 50)
x = t
y = np.sin(t)
dx = np.ones(50)
dy = np.cos(t)
fig15 = plt.figure(num=15)
ax15 = fig15.add_subplot(1, 1, 1)
ax15.quiver(x[::4], y[::4], dx[::4], dy[::4], units='xy',
 ↪  angles='xy', width=0.05, scale=2.0, label='Касательные
 ↪  векторы')
ax15.plot(x, y, linewidth=0.5, color='gray', label=r'$\sin(t)$')
ax15.set_xlabel(r'Ось $Ox$')
ax15.set_ylabel(r'Ось $Oy$')
ax15.legend(loc=1)
ax15.set_aspect('equal')
fig15.savefig('img15.pdf', format='pdf', bbox_inches='tight',
 ↪  pad_inches=0)
```

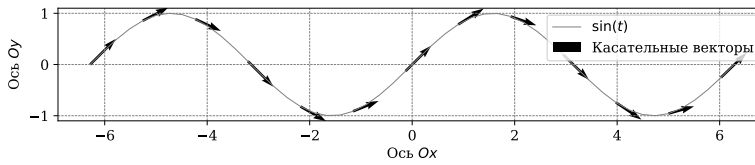

Рис. 6.5. Построение касательных векторов с помощью функции `quiver`

6.2.3.2. Ступенчатый график

При отображении дискретных данных часто есть необходимость в создании ступенчатых графиков. Например, если необходимо изобразить число заявок в некоторой системе обслуживания, то обычная функция `plot` с этой задачей не справится, так как будет пытаться соединить все точки прямыми отрезками, а не ступеньками. Этот факт иллюстрируется следующим примером (рис. 6.6):

```
fig16 = plt.figure(num=16, figsize=(10, 2))

ax16 = fig16.add_subplot(1, 1, 1)
ax16.step([1, 2, 3, 4, 5, 6, 7, 8], [0, 0, 1, 1, 2, 3, 1, 1],
 ↪  linewidth=1.0, label='Функция step')
ax16.plot([1, 2, 3, 4, 5, 6, 7, 8], [0, 0, 1, 1, 2, 3, 1, 1],
 ↪  linewidth=1.0, linestyle='--', label='Функция plot')
ax16.legend()

fig16.savefig('img16.pdf', format='pdf', bbox_inches='tight',
 ↪  pad_inches=0)
```

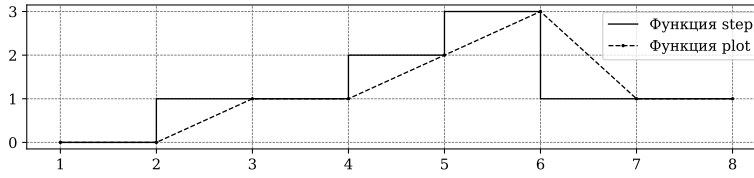

Рис. 6.6. Построение ступенчатого графика с помощью функции `step`

6.2.3.3. Горизонтальная и вертикальная прямые

Для изображения горизонтальной или вертикальной прямой можно обойтись и функцией `plot`, однако удобнее использовать специальные методы `hlines` и `vlines`. Пример ниже наглядно показывает, как это сделать (рис. 6.7):

```
x = np.linspace(0, 1.25, 100)
y = np.sqrt(x)

fig17 = plt.figure(num=16, figsize=(10, 2))
ax17 = fig17.add_subplot(1, 1, 1)
ax17.set(xlim=(0.0, 1.25), ylim=(0.0, 1.5))

ax17.plot(x, y)

ax17.plot([0.25, 1.0, 1.0], [0.5, 0.5, 1.0], linestyle='None',
 ↪  marker='o', markersize=5)

ax17.hlines(y=1.0, xmin=0.0, xmax=1.0, linestyle='--', lw=1)
ax17.hlines(y=0.5, xmin=0.0, xmax=1.0, linestyle='--', lw=1)
ax17.vlines(x=0.25, ymin=0.0, ymax=0.5, linestyle='--', lw=1)
ax17.vlines(x=1.0, ymin=0.0, ymax=1.0, linestyle='--', lw=1)
```

```
ax17.text(0.6, 0.25, r'$\Delta x$', fontsize=16)
ax17.text(1.025, 0.65, r'$\Delta y = \sqrt{\Delta x}$',
 ↪  fontsize=16)

fig17.savefig('img17.pdf', format='pdf', bbox_inches='tight',
 ↪  pad_inches=0)
```

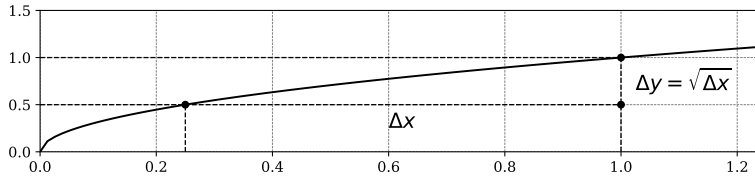

Рис. 6.7. Построение горизонтальной и вертикальной кривых

6.2.4. Создание анимации с помощью ffmpeg

Рассмотрим способ создания анимированных кривых с помощью библиотеки `Matplotlib` и утилиты `ffmpeg`. Несмотря на то что в `Matplotlib` встроены собственные средства для создания анимации, подход с использованием ffmpeg более универсален, так как его можно использовать в связке не только с `Matplotlib` но и с любыми другими библиотеками и утилитами для визуализации графиков.

Программа `ffmpeg` представляет собой утилиту, способную кодировать и декодировать видеофайлы, а также создавать видео из некоторого числа растровых изображений. Мы рассмотрим способ создания видеофайла в формате `.mp4` из набора `.png` файлов, которые будут генерироваться с помощью библиотеки `Matplotlib`.

Утилиту `ffmpeg` можно скачать на официальном сайте `https://www.ffmpeg.org/download.html`. Доступны сборки практически для всех операционных систем. Утилита представляет собой один исполняемый файл. В данном примере рассмотрим запуск `ffmpeg` под операционной системы Windows, однако он с минимальными изменениями может быть перенесён на случай Unix/GNU Linux и MacOS.

Кроме стандартных импортов библиотеки `NumPy` и `Matplotlib` необходимо импортировать подмодуль `mpatches`, который мы будем использовать для отрисовки окружности, и два модуля стандартной библиотеки — `os` и `subprocess`. Модуль `os` позволяет получить доступ к стандартным операционной системы. В данном случае нам понадобится функция `mkdir`, создающая пустую директорию. Модуль `subprocess` служит для фонового запуска внешних программ. Мы используем его для запуска `ffmpeg` непосредственно из `Python`-программы. Список импортов, таким образом, будет выглядеть следующим образом:

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

import subprocess
import os
```

В качестве примера рассмотрим анимацию циклоиды, которую описывает зафиксированная точка на катящейся окружности. Начнём с того, что создадим параметрическую функцию, задающую точки циклоиды:

$$\mathbf{r}(t) = \begin{cases} R(t - \sin t), \\ R(1 - \cos t), \end{cases}$$

где R — радиус генерирующей циклоиду окружности, а t — параметр, принимающий значения из $\mathbb{R}$.

```
def cycloid(t, R=3.0):
    '''Уравнение циклоиды'''
    return np.array([R*(t - np.sin(t)), R*(1 - np.cos(t))])

# укажем директорию, в которую будем
# сохранять сгенерированные картинки
FOLDER = 'cycloid'
try:
    os.mkdir(FOLDER)
except FileExistsError:
    pass

fig18 = plt.figure(1)
ax01 = fig18.add_subplot(1, 1, 1)
ax01.set_aspect('equal')
# радиус генерирующей окружности
R = 2.0
final_t = 10
final_x, _ = cycloid(final_t, R=R)

for counter, last_t in enumerate(np.arange(0.0, final_t + 0.1,
 ↪  0.1)):
    # стираем все, что было на картинке
    ax01.clear()

    T = np.linspace(0.0, last_t, 1000)
    X, Y = cycloid(T, R=R)
    # последняя точка кривой
    last_x, last_y = cycloid(T[-1], R=R)

    # Генерирующая окружность с центром в точке (Rt, R)
    circ = mpatches.Circle((R*T[-1], R), R, fill=False,
     ↪  color='black')
    # Точка на катящейся окружности
    last_point = mpatches.Circle((last_x, last_y), 0.1,
     ↪  color='black')
    # задаем границы осей
    ax01.set_xlim(right=final_x + 2*R)
    ax01.set_ylim(top=2*R)
    # рисуем циклоиду
    ax01.plot(X, Y, linestyle='--')
    # добавляем на картинку точку и окружность
    ax01.add_patch(last_point)
```

```
    ax01.add_patch(circ)

    fig18.savefig('{0}/{1:03d}.png'.format(FOLDER, counter),
     ↪  dpi=300, format='png')
```

После завершения работы программы, будет создана директория `cycloid`, куда будут сохранены все сгенерированные картинки. Для сборки картинок в видеофайл следует запустить `ffmpeg` в консоли со следующими опциями:

```
/usr/bin/ffmpeg -y -r 30 -f image2 -i cycloid\%03d.png -vcodec
 ↪  libx264 -crf 25 -pix_fmt yuv420p cycloid.mp4
```

Здесь указан полный путь до исполняемого файла `ffmpeg`, который зависит от способа установки и может отличаться на разных компьютерах. Все остальные опции следует указать без изменения вне зависимости от операционной системы. Не будем вдаваться в подробности опций, укажем лишь, что `cycloid%03d.png` обозначает файлы в директории `cycloid` с именами из трёх цифр: начиная от `000.png` и заканчивая `999.png`. В качестве последнего аргумента указано имя создаваемого видеофайла `cycloid`.

Следующий пример показывает, как запустить `ffmpeg` непосредственно из `Python`-программы, не открывая консоль:

```
FFMPEG = '/usr/bin/ffmpeg'
CMD = [FFMPEG, '-y', '-r', '30', '-f', 'image2', '-i',
 ↪  '{0}\%03d.png'.format(FOLDER), '-vcodec', 'libx264', '-crf',
 ↪  '25', '-pix_fmt', 'yuv420p', 'cycloid.mp4']
subprocess.run(CMD)
```

В результате будет создан видеофайл `cycloid.mp4` с анимацией катящейся окружности, точка которой отрисовывает линию циклоиды. Предпоследний кадр изображён на рис. 6.8.

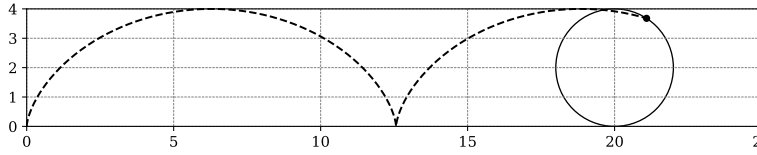

Рис. 6.8. Предпоследний кадр отрисовки циклоиды

6.3. Задание

1. Постройте изображения основных графических примитивов и модифицированные графики из примеров раздела 6.2 (см. раздел 6.4.1).
2. Постройте анимированные изображения, базируясь на примерах раздела 6.2 (см. раздел 6.4.2).

6.4. Порядок выполнения лабораторной работы

Так же как в предыдущей лабораторной работе, большинство заданий ориентированы на модификацию примеров из раздела 6.2. Перед внесением изменений рекомендуется запустить исходный код примеров и убедиться, что получившиеся

изображения совпадают с теми, которые приведены в данном пособии, и только после этого приступать к модификации кода.

6.4.1. Построение изображений с использованием расширенных возможностей Matplotlib

1. Взяв за образец пример `fig11`, постройте изображения основных графических примитивов, доступных в `Matplotlib`. Измените параметры всех фигур так, чтобы они отличались от изображённых на рис. 6.1:
   - измените цвет и радиус окружности;
   - измените полуоси и цвет эллипса;
   - измените угол наклона прямоугольника;
   - измените арку эллипса так, чтобы она соответствовала эллипсу с полуосями 2 и 1;
   - поверните сегмент окружности на 90 градусов вокруг центра;
   - с помощью ломаной линии постройте трапецию;
   - постройте красную стрелку, направленную вправо;
   - постройте три прямоугольника со скруглёнными краями, вложенных один в другой.
2. Используя пример `fig12`, с помощью функций `mlines.Line2D` и `np.sin` постройте график синуса. Не используйте при этом функцию `plot`.
3. Используя примеры `fig13` и `fig14`, постройте рисунок, аналогичный рис. 6.3, но используйте не метод `text`, а метод `annotate`.
4. Измените пример `fig15` так, чтобы вместо касательных векторов для графика синуса он строил касательные вектора для окружности.

6.4.2. Построение анимированных изображений

Изучите пример из пункта 6.2.4 построения анимированной циклоиды.

1. Скачайте с официального сайта [2] утилиты `ffmpeg` необходимые исполняемые файлы.
2. Запустите пример и проверьте его работоспособность.
3. Проверьте, правильно ли указан путь к исполняемому файлу `ffmpeg`.
4. Устраните все ошибки (если они возникнут), описав в отчёте, что конкретно вы исправляли.
5. Измените пример так, чтобы циклоида строилась в обратном порядке (окружность катилась справа налево).

В качестве справочной литературы рекомендуется использовать следующие источники [1; 2; 4; 5].

6.4.3. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:

- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку цели работы;
- описание результатов выполнения задания:
  - полученные вами графические изображения;
  - листинги изменённых вами примеров с указанием того, какие части примеров были вами изменены и с какой целью это было сделано;
- выводы, согласованные с целью работы.

К отчёту необходимо приложить файл с анимацией циклоиды.

6.5. Контрольные вопросы

1. Что такое примитивы?
2. Какие примитивы есть в `Matplotlib`?
3. Какой метод позволяет добавить созданные примитивы на график?
4. Для чего нужен метод `Line2D`?
5. Какой метод позволяет добавлять на координатную плоскость текстовые метки?
6. Какие параметры, регулирующие внешний вид текста на координатной плоскости, вы знаете?
7. Чем аннотации отличаются от обычного текста?
8. Может ли метод `annotate` заменить метод `text`?
9. Как настраиваются элементы аннотаций?
10. Какая функция позволяет визуализировать векторное поле?
11. Какая функция позволяет построить ступенчатый график?
12. Как в `Matplotlib` проще всего построить горизонтальную и вертикальную прямые линии?
13. Для чего нужна утилита `ffmpeg` и как её использовать в связке с `Matplotlib`?
14. Пользуясь дополнительными источниками и официальной документации, ответьте на следующие вопросы:
    - Какая функция `Matplotlib` позволяет строить гистограммы?
    - Как установить логарифмический масштаб вдоль одной из осей координат?
    - Как минимизировать поля при сохранении созданного изображения в файл?
    - Какие функции `Matplotlib` позволяют строить трёхмерные графики?
    - Есть ли в `Matplotlib` встроенные средства создания анимированных графиков?
    - Какие недостатки `Matplotlib` вы можете выделить после знакомства с этой библиотекой?

Список литературы

1. *Любанович Б.* Простой Python. Современный стиль программирования. — М. : Питер, 2017. — ISBN 978-5-496-02088-6.
2. FFmpeg home site. — 2018. — URL: `http://www.ffmpeg.org`.
3. Matplotlib home site. — 2018. — URL: `https://matplotlib.org/`.
4. Python home site. — 2018. — URL: `https://www.python.org/`.
5. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 7. Построение кривых на плоскости

7.1. Цель лабораторной работы

Целью данной лабораторной работы является освоение комплексного применения библиотеки NumPy и Matplotlib, а также базовых средств языка Python для построения кривых на плоскости и реализации анимированных изображений геометрических преобразований на плоскости.

7.2. Предварительные сведения

7.2.1. Компьютерная геометрия на плоскости

Компьютерная геометрия — это раздел прикладной математики, в котором изучается и разрабатывается математический аппарат, используемый в компьютерной (машинной) графике [1—3]. Основной задачей компьютерной геометрии является разработка способов эффективного вычисления точек плоских и пространственных кривых, а также поверхностей в трёхмерном пространстве. Также изучаются различные геометрические преобразования на плоскости и в пространстве, такие как вращение, перенос, изменение масштаба и т.д.

Результаты компьютерной геометрии используются в самых разных областях человеческой деятельности, связанной с применением современных компьютеров. В качестве примеров можно привести системы автоматизированного проектирования (CAD — computer aided design, CAM — computer aided manufacturing), векторные графические редакторы (Adobe Illustrator, CorelDRAW, Inkscape), векторные форматы изображений (SVG, PDF, PS), издательское дело (векторные шрифты, например, стандарты TrueType и OpenType), веб-программирование (технология Canvas), компьютерные игры (DirectX и OpenGL) и компьютерная трёхмерная анимация, которая в настоящее время сильно потеснила классическую рисованную анимацию.

7.2.2. Пространственные и плоские кривые

В данном разделе мы кратко изложим сведения из аналитической и дифференциальной геометрии, необходимые для понимания дальнейшего материала. За более подробным изложением следует обратиться к учебным пособиям по этому предмету [4—11].

Геометрическое место точек, радиус-векторы $\mathbf{r}$ которых определяются уравнением

$$\mathbf{r} = \mathbf{r}(t), \quad t \in [a, b] \subset \mathbb{R},$$

представляет *кривую* (или, более точно, регулярную часть кривой), если функция $\mathbf{r}(t)$ непрерывна и имеет непрерывную производную $\dot{\mathbf{r}}(t)$, не обращающуюся внутри отрезка $[a, b]$ в ноль:

$$\frac{r}{t} \neq 0, \quad \forall t \in [a, b].$$

В евклидовом пространстве E^n с базисом $\langle \mathbf{e}_1, \ldots, \mathbf{e}_n \rangle$ кривая представима в виде системы из n непрерывных и дифференцируемых функций аргумента t:

$$\mathbf{r} = \sum_{i=1}^{n} x^i(t)\mathbf{e}_i = x^1(t)\mathbf{e}_1 + \ldots + x^n(t)\mathbf{e}_n.$$

В частности, в трёхмерном евклидовом пространстве в декартовых координатах $\mathbf{r}(t) = x(t)\mathbf{e}_1 + y(t)\mathbf{e}_2 + z(t)\mathbf{e}_3$, где $\mathbf{e}_1 = (1,0,0)^T$, $\mathbf{e}_2 = (0,1,0)^T$, $\mathbf{e}_3 = (0,0,1)^T$,

$$\mathbf{r}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix}.$$

Производная от векторозначной функции $\mathbf{r}(t)$ в произвольной точке

$$\frac{d\mathbf{r}}{dt} = \left(\frac{dx^1}{dt}, \dots, \frac{dx^n}{dt} \right)^T = (\dot{x}^1, \dots, \dot{x}^n)^T$$

называется *касательным вектором* и устанавливает положительное направление касательной в точке P к кривой, задаваемой функцией $\mathbf{r}$.

Одну и ту же кривую можно параметризовать разными способами. Рассмотрим параметр $l = l(t)$, называемый *натуральным*, при котором касательный вектор, получающийся при дифференцировании функции по этому параметру, будет единичным вектором при любых значениях l:

$$\left| \frac{d\mathbf{r}}{dl} \right| \equiv 1, \quad \forall l \in [a, b].$$

Чтобы найти связь l и t, продифференцируем $\mathbf{r}(l)$ как сложную функцию $\mathbf{r}(l(t))$:

$$\frac{d\mathbf{r}(l(t))}{dt} = \frac{d\mathbf{r}}{dl}\frac{dl}{dt} \Rightarrow \left| \frac{d\mathbf{r}}{dt} \right| = \left| \frac{d\mathbf{r}}{dl}\frac{dl}{dt} \right| = \frac{dl}{dt}\left| \frac{d\mathbf{r}}{dl} \right| = \frac{dl}{dt}.$$

В результате найдём связь между dt и dl:

$$dl = \|\dot{\mathbf{r}}(t)\|dt = \left\| \frac{d\mathbf{r}}{dt} \right\| dt = \sqrt{\left(\frac{d\mathbf{r}}{dt}, \frac{d\mathbf{r}}{dt} \right)} dt.$$

В частности, в случае пространства E^3 можно записать

$$dl = \sqrt{\dot{x}^2(t) + \dot{y}^2(t) + \dot{z}^2(t)}dt.$$

Геометрический смысл натурального параметра l заключается в следующем. Мы фиксируем на кривой некоторую точку O и принимаем её за начало отсчёта. Любую другую точку P кривой можно однозначно определить как расстояние l, пройденное по кривой от точки O до точки P. Положительные и отрицательные значение l соответствуют разным направлениям перемещения по кривой. Заметим, что если удастся проинтегрировать выражение $\|\dot{\mathbf{r}}\|dt$, то будет найдена связь между t и l с точностью до константы. Константа как раз отражает произвол в выборе точки начала отсчёта.

Кривая, которая допускает введение натурального параметра, и, как следствие, длины дуги, называется *спрямляемой*. Кривая спрямляема, если текущие координаты являются непрерывными функциями параметра t с непрерывными первыми производными. Натуральный параметр l на кривой, для которого производная от

радиус-вектора становится единичным вектором, есть длина дуги кривой, измеряемая от произвольного, но определённо выбранного начала отсчёта дуги на кривой:

$$l = \int_c^t \sqrt{\dot{x}^2 + \dot{y}^2 + \dot{z}^2}\,dt.$$

7.3. Задание

1. Построить график функции $y(x) = x^2 + x + 1$, $x = -4, \ldots, 4$, оформив его по аналогии с рис. 7.1, т.е. добавив аннотации, обозначения осей Ox и Oy, заголовок, содержащий формулу функции.

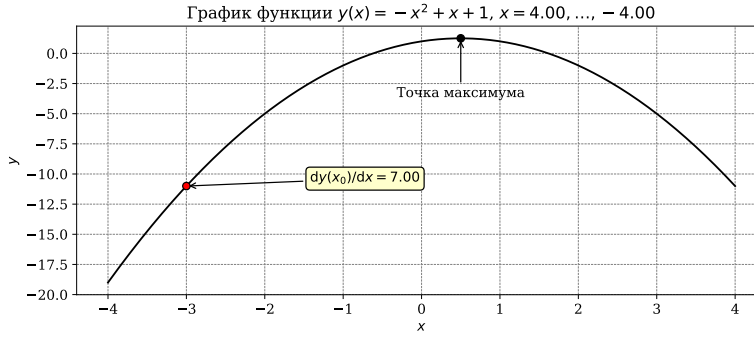

Рис. 7.1. График функции $y(x) = x^2 + x + 1$

2. Построить график функции распределения простых чисел (см. раздел 7.4.1).
3. Построить графики параметрических кривых эллипса и гиперболы. Дополнительный указания см. в разделе 7.4.2.
4. Построить вектор, начинающийся в точке $(0, 0; 0, 0)$. Реализуйте анимацию вращения вектора по и против часовой стрелки. Дополнительный указания см. в разделе 7.4.3.

7.4. Порядок выполнения лабораторной работы

7.4.1. Построение функции распределения

Требуется построить график функции $\pi(x)$, которая задаёт распределение простых чисел. Значения $\pi(x)$ равны количеству простых чисел, меньших либо равных действительному числу x.

1. Скачайте со страницы `https://primes.utm.edu/lists/small/millions/` файл с первым миллионом простых чисел.
2. Для тестирования вашей реализации функции $\pi(x)$ используйте значения функции $\pi(x)$ для x, равных степени десятки 10^n при n от 0 до 26 (см. `http://oeis.org/A006880/list`).

3. Для считывания простых чисел из файла можно использовать функции `loadtxt` или `genfromtxt` библиотеки `NumPy`. При этом начать считывать нужно с третьей строки, так как в первой строке содержится текстовое описание, а вторая строка служит разделителем и поэтому пустая. Кроме того, при считывании следует преобразовать данные в целый тип (опция `dtype`). Числа в файле расположены в несколько колонок, создавая таким образом при считывании многомерный массив, который необходимо преобразовать в одномерный массив. Для этого используйте метод `flatten`.
4. Для построения графика используйте функцию `step` библиотеки `Matplotlib`. Постройте точечный график для 60 первых чисел и линейчатый график для 200. Изображения должны соответствовать приведённым на рис. 7.2.

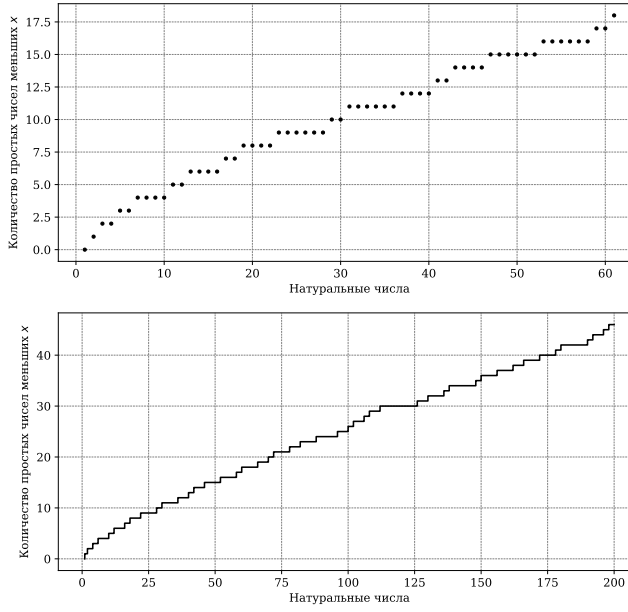

Рис. 7.2. Графики функции распределения простых чисел

7.4.2. Построение параметрических кривых

1. Требуется построить графики параметрических кривых, а именно эллипса и гиперболы с несовпадающими центрами (x_0, y_0), но с одинаковыми параметрами a и b (рис. 7.3).
   Для построения используйте следующие параметрические уравнения:

$$\text{эллипс: } \begin{cases} x = x_0 + a\cos t, \\ y = y_0 + b\sin t, \\ t \in [0, 2\pi], \end{cases} \qquad \text{гипербола: } \begin{cases} x = \pm(x_0 + a\ \mathrm{ch}\ t), \\ y = y_0 + b\ \mathrm{sh}\ t, \\ t \in (-\infty, \infty). \end{cases}$$

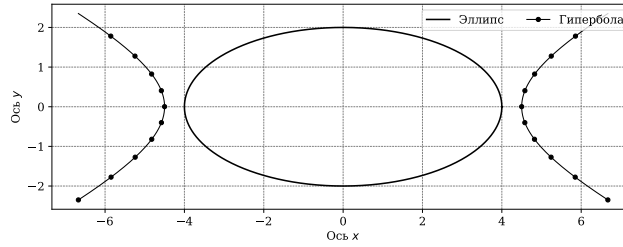

Рис. 7.3. Эллипс и гипербола с несовпадающими центрами

2. Семейство эллипсов и гипербол образует координатные линии так называемой «эллиптической системы координат», которую чаще всего задают следующим образом:

$$\begin{cases} x = a \operatorname{ch} \mu \cos \nu, \\ y = b \operatorname{sh} \mu \sin \nu, \end{cases} \quad \mu \in \mathbb{R}_+, \quad \nu \in [0, 2\pi].$$

Координатные линии такой системы координат состоят из конфокальных эллипсов и гипербол. Постройте такую систему, взяв за образец рис. 7.4.

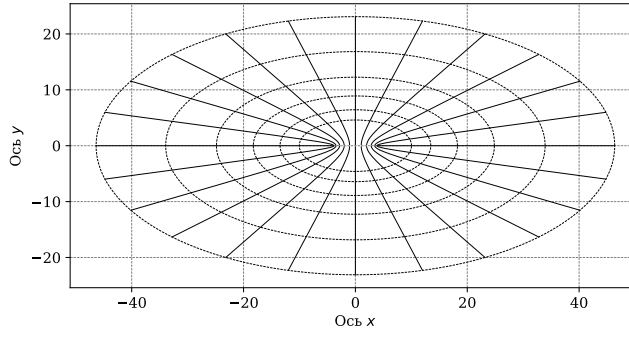

Рис. 7.4. Эллиптическая система координат

7.4.3. Построение векторов

Требуется изобразить вектор, исходящий из начала координат и реализовать анимацию его вращения по и против часовой стрелки (см. рис. 7.5).

Для того, чтобы изображение не изменялось в размере по мере вращения вектора, зафиксируйте максимальное и минимальное значения по осям Ox и Oy. Для создания анимации можно применить утилиту `ffmpeg` или встроенные возможности библиотеки `Matplotlib`.

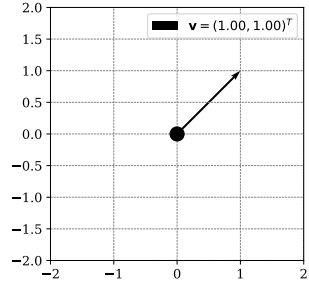

Рис. 7.5. Изображение вектора

7.4.4. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:

- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку задания;
- описание результатов выполнения задания:
  - полученные вами графические изображения;
  - листинги (исходный код) программ (если они есть);
- выводы, согласованные с целью работы.

К отчёту необходимо приложить файл с анимацией.

7.5. Контрольные вопросы

1. Что изучает компьютерная геометрия? Чем она отличается от компьютерной (машинной) графики?
2. Математический аппарат каких теоретических разделов математики используется в компьютерной геометрии?
3. Дайте определение плоской и трехмерной кривой, которое используется в компьютерной геометрии.
4. Какие три основных способа записи уравнений, задающих кривую на плоскости и в пространстве? Какой из них наиболее часто используется при построении кривых с помощью компьютера?
5. Что такое натуральный параметр кривой? Как его найти? Каков его геометрический смысл?

Список литературы

1. *Роджерс Д.*, *Адамс А.* Математические основы машинной графики. — М. : Мир, 2001. — ISBN 5030021434.
2. *Голованов Н. Н.* Геометрическое моделирование. — М. : Физматлит, 2002. — ISBN 5940520480.
3. *Никулин Е. А.* Компьютерная геометрия и алгоритмы машинной графики. — Санкт-Петербург : БХВ-Петербург, 2003.

4. *Дубровин Б. А.*, *Новиков С. П.*, *Фоменко А. Т.* Современная геометрия: Методы и приложения. Т. 1. — 6-е изд. — М. : УРСС: Книжный дом «ЛИБРОКОМ», 2013. — ISBN 978-5-453-00047-0.
5. *Дубровин Б. А.*, *Новиков С. П.*, *Фоменко А. Т.* Современная геометрия: Методы и приложения. Т. 2. — 6-е изд. — М. : УРСС: Книжный дом «ЛИБРОКОМ», 2013. — ISBN 978-5-453-00048-7.
6. *Рашевский П. К.* Риманова геометрия и тензорный анализ. Т. 1. — М. : УРСС, 2014. — ISBN 978-5-396-00577-8.
7. *Рашевский П. К.* Риманова геометрия и тензорный анализ. Т. 2. — М. : УРСС, 2014. — ISBN 978-5-396-00578-5.
8. *Мищенко А. С.*, *Фоменко А. Т.* Краткий курс дифференциальной геометрии и топологии. — М. : ФИЗМАТЛИТ, 2017. — ISBN 978-5-9710-2681-5.
9. *Фиников С. П.* Курс дифференциальной геометрии. — М. : URSS, 2017.
10. *Шаров Г. С.*, *Шелехов А. М.*, *Шестакова М. А.* Дифференциальная геометрия и топология в задачах. — М. : Ленанд, 2017. — ISBN 978-5-9710-3743-9.
11. *Щепетилов А. В.* Введение в дифференциальную геометрию. — М. : КДУ, 2017. — ISBN 978-5-91304-710-6.

Лабораторная работа № 8. Геометрические преобразования на плоскости. Репер Френе

8.1. Цель лабораторной работы

Целью данной лабораторной работы выработка навыков построения различных кривых линий и визуализации функций с помощью изученных ранее библиотек.

8.2. Предварительные сведения

Вторая производная от векторозначной функции $\mathbf{r}(l)$ в точке $P = \mathbf{r}(l_0) = (x^1(l_0), \dots, x^n(l_0))$

$$\frac{d^2\mathbf{r}}{dl^2}(l_0) = \ddot{\mathbf{r}}(t) = (\ddot{x}^1(l_0), \dots, \ddot{x}^n(l_0)),$$

называется *вектором нормали* и устанавливает положительное направление нормали в точке P к кривой, задаваемой функцией $\mathbf{r}(t)$.

Векторы касательной и нормали ортогональны, если параметр l — натуральный, что видно из следующих формул:

$$\frac{d}{dl}\left(\frac{d\mathbf{r}}{dl}, \frac{d\mathbf{r}}{dl}\right) = \left(\frac{d^2\mathbf{r}}{dl^2}, \frac{d\mathbf{r}}{dl}\right) + \left(\frac{d\mathbf{r}}{dl}, \frac{d^2\mathbf{r}}{dl^2}\right) = 2\left(\frac{d\mathbf{r}}{dl}, \frac{d^2\mathbf{r}}{dl^2}\right),$$

$$\frac{d}{dl}\left(\frac{d\mathbf{r}}{dl}, \frac{d\mathbf{r}}{dl}\right) = \frac{d}{dl}\left|\frac{d\mathbf{r}}{dl}\right|^2 = \frac{d}{dl}1 = 0,$$

$$\left(\frac{d\mathbf{r}}{dl}, \frac{d^2\mathbf{r}}{dl^2}\right) = 0.$$

Кривизной кривой $\mathbf{r}(l)$ называется длина вектора нормали k:

$$k = \left|\frac{d^2\mathbf{r}}{dl^2}\right|,$$

а *радиусом кривизны* называется число, обратное значению кривизны $R = 1/k$.

Обычно используют *единичный вектор нормали* $\mathbf{n}$, который определяется с помощью формулы

$$\frac{d^2\mathbf{r}}{dl^2} = k\mathbf{n}, \ \mathbf{n} = \frac{\dfrac{d^2\mathbf{r}}{dl^2}}{\left|\dfrac{d^2\mathbf{r}}{dl^2}\right|}.$$

Касательный вектор имеет физический смысл вектора скорости, а нормальный вектор — вектора ускорения.

Вектор $\mathbf{b} = [\mathbf{v}, \mathbf{n}]$ называется вектором *бинормали*. По определению векторного умножения вектор бинормали ортогонален векторам $\mathbf{v}$ и $\mathbf{n}$. Тройка векторов $\langle \mathbf{v}, \mathbf{n}, \mathbf{b} \rangle$ образует репер, который называется *репером Френе*:

$$[\mathbf{v}, \mathbf{n}] = \mathbf{b}, \ [\mathbf{n}, \mathbf{b}] = \mathbf{v}, \ [\mathbf{b}, \mathbf{v}] = \mathbf{n}.$$

Для любой пространственной кривой $\mathbf{r}(l)$, где l — натуральный параметр, имеют место следующие формулы, называемые *формулами Френе*:

$$\frac{d\mathbf{v}}{dl} = k\mathbf{n},\ \frac{d\mathbf{n}}{dl} = -k\mathbf{v} - \kappa\mathbf{b},\ \frac{d\mathbf{b}}{dl} = \kappa\mathbf{n},$$

где $\mathbf{v}$ — вектор касательной, $\mathbf{n}$ — вектор нормали, $\mathbf{b}$ — вектор бинормали, κ — *кручение*, а k — кривизна.

В случае плоской кривой репер Френе состоит только из двух векторов: касательного и нормального.

Приведём формулы для компонент этих векторов в декартовых координатах для плоской кривой общего вида. Плоская кривая в декартовой системе координат в параметрическом виде задаётся следующей формулой:

$$\mathbf{r}(t) = (x(t), y(t))^T,$$

а касательный и нормальный векторы вычисляются следующим образом:

$$\mathbf{v} = \frac{d\mathbf{r}}{dt} = (\dot{x}(t), \dot{y}(t)),\ \mathbf{n} = \frac{d^2\mathbf{r}}{dt^2} = (\ddot{x}(t), \ddot{y}(t)).$$

Репер Френе образуют нормированные векторы:

$$\mathbf{v}_0 = \frac{\mathbf{v}}{|\mathbf{v}|} = \left(\frac{\dot{x}}{\sqrt{\dot{x}^2+\dot{y}^2}}, \frac{\dot{y}}{\sqrt{\dot{x}^2+\dot{y}^2}}\right),\ \mathbf{n}_0 = \frac{d\mathbf{v}_0/dt}{|d\mathbf{v}_0/dt|}.$$

Следует обратить внимание на то, что для вычисления нормированного нормального вектора надо взять первую производную от нормированного вектора $\mathbf{v}_0$, а не вторую производную от радиус-вектора $\mathbf{r}(t)$.

Явное выражение компонент нормального вектора через функции $x(t)$ и $y(t)$ имеет простой вид:

$$\frac{d}{dt}\frac{\dot{x}}{\sqrt{\dot{x}^2+\dot{y}^2}} = \frac{\dot{y}(\dot{y}\ddot{x} - \dot{x}\ddot{y})}{(\sqrt{\dot{x}^2+\dot{y}^2})^3},\ \frac{d}{dt}\frac{\dot{y}}{\sqrt{\dot{x}^2+\dot{y}^2}} = \frac{\dot{x}(\dot{x}\ddot{y} - \dot{y}\ddot{x})}{(\sqrt{\dot{x}^2+\dot{y}^2})^3},\ \left|\frac{d\mathbf{v}_0}{dt}\right| = \frac{\dot{x}\ddot{y} - \dot{y}\ddot{x}}{\dot{x}^2+\dot{y}^2},$$

$$\mathbf{n}_0 = \frac{d\mathbf{v}_0}{dt} : \left|\frac{d\mathbf{v}_0}{dt}\right| = \left(\frac{-\dot{y}}{\sqrt{\dot{x}^2+\dot{y}^2}}, \frac{\dot{x}}{\sqrt{\dot{x}^2+\dot{y}^2}}\right).$$

Эволютой кривой называют геометрическое место точек, являющихся центрами кривизны данной кривой. *Эвольвентой* (инволютой) кривой в свою очередь называется кривая, нормаль в каждой точки которой является касательной к исходной кривой. По отношению к своей эволюте любая кривая является эвольвентой.

Зная явные формулы для касательного и нормального векторов в декартовой системе координат, можно получить явные формулы для радиус-вектора эволюты. Центр кривизны $\mathbf{M}$ лежит на главной нормали на расстоянии радиуса кривизны от кривой:

$$\mathbf{M} = \mathbf{r} + R\mathbf{n}.$$

Радиус кривизны в декартовой системе координат вычисляется по формуле

$$R = \frac{(\dot{x}^2+\dot{y}^2)^{3/2}}{\dot{x}\ddot{y} - \dot{y}\ddot{x}}.$$

При этом радиус-вектор эволюты имеет вид

$$\mathbf{M} = \begin{pmatrix} x - \dot{y}\dfrac{\dot{x}^2 + \dot{y}^2}{\dot{x}\ddot{y} - \dot{y}\ddot{x}} \\ y - \dot{x}\dfrac{\dot{x}^2 + \dot{y}^2}{\dot{x}\ddot{y} - \dot{y}\ddot{x}} \end{pmatrix}.$$

8.3. Задание

1. Построить гипоциклоиду, отобразив на графике большую (внешнюю) окружность, несколько малых (внутренних) окружностей, центры и радиусы малых окружностей. Дополнительные указания см. в разделе 8.4. Создать анимацию построения гипоциклоиды, анимировав малую окружность, катящуюся по внутренней стороне большей окружности.
2. Построить некоторую двумерную кривую, а также репер Френе (касательный и нормальный векторы) в произвольной точке этой кривой (рис. 8.1). Создать анимацию перемещения репера вдоль кривой. Масштаб осей должен быть одинаковым, иначе векторы репера будут выглядеть неортогональными.
3. Найти эволюту эллипса, астроиды и циклоиды. Изобразить каждую кривую и её эволюту (см. рис. 8.2), предполагая, что первая и вторая производные радиус-вектора вычислены аналитически и программе их вычислять не придётся. Уравнения астроиды и циклоиды соответственно имеют вид

$$\mathbf{r}(t) = \begin{bmatrix} R\cos^3 t \\ R\sin^3 t \end{bmatrix}, \qquad \mathbf{r} = \begin{bmatrix} R(t - \sin t) \\ R(1 - \cos t) \end{bmatrix}.$$

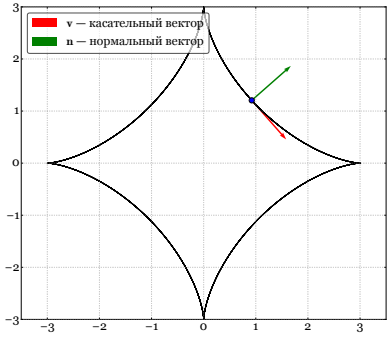

Рис. 8.1. Двумерная кривая и её репер Френе в некоторой точке

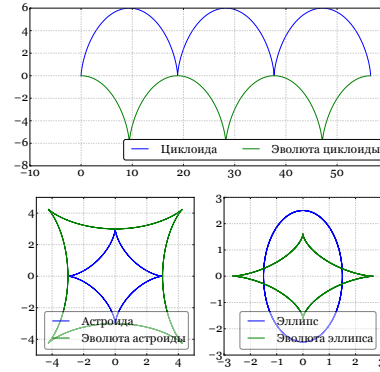

Рис. 8.2. Эллипс, астроида, циклоида и их эволюты

8.4. Порядок выполнения задания по построению гиперциклоиды

Уравнение гипоциклоиды имеет вид

$$x(t) = (R - r)\cos t + r\cos\left(\frac{R - r}{r}t\right),$$
$$y(t) = (R - r)\sin t - r\sin\left(\frac{R - r}{r}t\right).$$

Для отображения на графике большой (внешней) окружности, нескольких малых (внутренних) окружностей, центров и радиусов малых окружностей изменяйте параметр $k = \frac{R}{r}$, последовательно присваивая ему целые, дробно-рациональные и иррациональные значения (рис. 8.3).

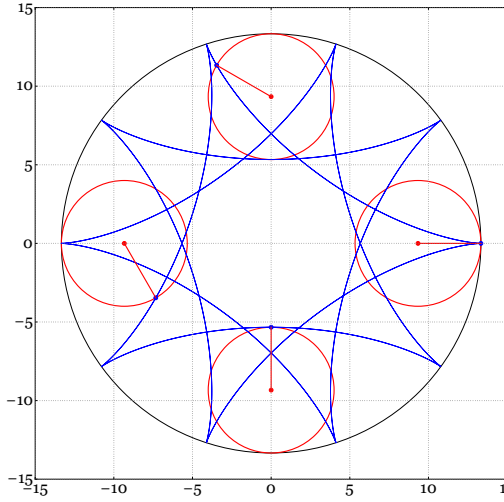

Рис. 8.3. Гиперциклоида с разными значениями параметра $k = \frac{R}{r}$

Как меняется поведение кривой для этих трёх случаев? Для ответа на этот вопрос постройте разные изображения гипоциклоиды, варьируя диапазон изменения параметра t.

В качестве справочной литературы рекомендуется использовать следующие источники [1—6].

8.4.1. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:

– титульный лист с указанием номера лабораторной работы и ФИО студента;
– формулировку задания;

– описание результатов выполнения задания:
  – полученные вами графические изображения;
  – листинги (исходный код) программ (если они есть);
– выводы, согласованные с целью работы.

К отчёту необходимо приложить архив с файлами анимации.

8.5. Контрольные вопросы

1. Какие векторы составляют репер Френе в двумерном и трехмерном случае?
2. Что такое кривизна кривой? Что такое центр кривизны? Радиус кривизны?
3. Что такое резольвента и эволюта? Как они связаны?

Список литературы

1. *Роджерс Д.*, *Адамс А.* Математические основы машинной графики. — М. : Мир, 2001. — ISBN 5030021434.
2. *Голованов Н. Н.* Геометрическое моделирование. — М. : Физматлит, 2002. — ISBN 5940520480.
3. *Дубровин Б. А.*, *Новиков С. П.*, *Фоменко А. Т.* Современная геометрия: Методы и приложения. Т. 1. — 6-е изд. — М. : УРСС: Книжный дом «ЛИБРОКОМ», 2013. — ISBN 978-5-453-00047-0.
4. *Фиников С. П.* Курс дифференциальной геометрии. — М. : URSS, 2017.
5. Python home site. — 2018. — URL: `https://www.python.org/`.
6. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 9. Интерполяция. Феномен Рунге

9.1. Цель лабораторной работы

Целью данной лабораторной работы является знакомство с интерполяцией с помощью сплайнов.

9.2. Предварительные сведения

Во многих практических задачах аналитическая форма некоторой двумерной или трёхмерной кривой неизвестна и её необходимо восстановить по конечному набору точек и ряду дополнительных условий, налагаемых на искомую кривую в этих точках. Данная задача называется задачей *интерполяции кривой* по заданным точкам. Наиболее очевидный подход к этой задаче — соединить известные точки отрезками прямых и получить некоторую *ломаную линию*. Однако чаще всего полученная ломаная плохо отражает свойства искомой кривой и используется лишь для грубого приближения.

Более точные результаты даёт интерполяция с помощью *сплайнов*. Сплайном изначально называлась гибкая линейка, с помощью которой проводили соединительные кривые при построении чертежей на бумаге. Математический же сплайн — это кусочный полином степени n.

На сплайн часто налагают дополнительные условия типа условия существования первых и вторых производных в точках соединения сегментов. Степень n обычно мала (от 2 до 5), так как для полиномов высоких степеней вероятно возникновение *феномена Рунге* — возрастание погрешности интерполяции в окрестности концов промежутка интерполяции.

Задача ставится следующим образом: имеется совокупность точек на плоскости или в пространстве, радиус-векторы которых равны $\mathbf{p}_i = (\mathbf{p}_1, \dots, \mathbf{p}_n)$, $i = 0, \dots, n$. Требуется построить линию, радиус-вектор которой при значениях параметра t_i был бы равен $\mathbf{p}_i$:

$$\mathbf{r}(t_0) = \mathbf{p}_0, \dots, \mathbf{r}(t_i) = \mathbf{p}_i, \dots, \mathbf{r}(t_n) = \mathbf{p}_n.$$

Точки $\mathbf{p}_i$ называются *опорными* (или *характеристическими*) точками кривой, а значения параметра t_i — *узловыми* точками.

Кроме самих точек могут быть заданы производные первого $\dot{\mathbf{p}}_i = \dot{\mathbf{r}}(t_i)$ и второго порядка $\ddot{\mathbf{p}}_i = \ddot{\mathbf{r}}(t_i)$.

9.2.1. Ломаная

Простейшая заданная точечно линия называется *ломаной линией*. Она состоит из отрезков, последовательно соединяющих заданные точки. Значения параметра в каждой последующей точке больше предыдущего: $t_i < t_{i+1}$.

Радиус вектор ломаной определяется равенством

$$\mathbf{r}(t) = \mathbf{p}_i(1 - \tau) + \mathbf{p}_{i+1}\tau, \quad \tau = \frac{t - t_i}{t_{i+1} - t_i}, \quad t_0 \leqslant t \leqslant t_n.$$

Параметр τ называется внутренним, и он по определению нормирован так, что изменяется в пределах отрезка $[0, 1]$. Взяв две опорные точки $\mathbf{p}_i$ и $\mathbf{p}_{i+1}$ и последовательно изменяя параметр от $[0, 1]$, мы получим все точки отрезка прямой линии, соединяющей точки $\mathbf{p}_i$ и $\mathbf{p}_{i+1}$.

Большинство средств для построения графиков по заданным точкам фактически проводят интерполяцию с помощью ломаной линии. Если известно большое количество близко расположенных точек, то интерполяция ломаной может дать очень хорошее приближение к реальной кривой.

9.2.2. Интерполяция параметрическими полиномами высокого порядка

При использовании для интерполяции полиномов очевидным решением является выбор полинома максимально возможного порядка для достижения максимально возможной точности.

Пусть даны узловые точки $t_0, t_1, t_2, \ldots, t_{n-1}, t_n$ и соответствующие опорные точки $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, …, $\mathbf{p}_n$. Имеется возможность построить параметрический полином $\mathbf{r}(t)$ порядка не выше n:

$$\mathbf{r}(t) = \mathbf{a}_n t^n + \mathbf{a}_{n-1} t^{n-1} + \ldots + \mathbf{a}_1 t + \mathbf{a}_0.$$

Для нахождения коэффициентов полинома необходимо решить следующую систему уравнений:

$$\begin{cases} \mathbf{r}(t_0) = \mathbf{p}_0, \\ \mathbf{r}(t_1) = \mathbf{p}_1, \\ \qquad\vdots \\ \mathbf{r}(t_n) = \mathbf{p}_n, \end{cases}$$

которая в матричном виде записывается следующим образом:

$$\begin{pmatrix} 1 & t_0 & t_0^2 & \ldots & t_0^{n-1} & t_0^n \\ 1 & t_1 & t_1^2 & \ldots & t_1^{n-1} & t_1^n \\ 1 & t_2 & t_2^2 & \ldots & t_2^{n-1} & t_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & t_{n-1} & t_{n-1}^2 & \ldots & t_{n-1}^{n-1} & t_{n-1}^n \\ 1 & t_n & t_n^2 & \ldots & t_n^{n-1} & t_n^n \end{pmatrix} \begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_{n-1} \\ \mathbf{a}_n \end{pmatrix} = \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_{n-1} \\ \mathbf{p}_n \end{pmatrix},$$

где переменные и правая часть представляют собой матрицы размера $3 \times (n+1)$ (для плоской кривой размер матриц $2 \times (n+1)$):

$$\begin{pmatrix} \mathbf{a}_0 \\ \mathbf{a}_1 \\ \mathbf{a}_2 \\ \vdots \\ \mathbf{a}_{n-1} \\ \mathbf{a}_n \end{pmatrix} = \begin{pmatrix} a_0^x & a_0^y & a_0^z \\ a_1^x & a_1^y & a_1^z \\ a_2^x & a_2^y & a_2^z \\ \vdots & \vdots & \vdots \\ a_{n-1}^x & a_{n-1}^y & a_{n-1}^z \\ a_n^x & a_n^y & a_n^z \end{pmatrix}, \quad \begin{pmatrix} \mathbf{p}_0 \\ \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_{n-1} \\ \mathbf{p}_n \end{pmatrix} = \begin{pmatrix} p_0^x & p_0^y & p_0^z \\ p_1^x & p_1^y & p_1^z \\ p_2^x & p_2^y & p_2^z \\ \vdots & \vdots & \vdots \\ p_{n-1}^x & p_{n-1}^y & p_{n-1}^z \\ p_n^x & p_n^y & p_n^z \end{pmatrix}.$$

После нахождения коэффициентов $\{\mathbf{a}_i\}_0^n$ можно использовать параметрический полином для интерполяции искомой кривой. Однако, вопреки ожиданиям, увеличение порядка полинома необязательно приводит к повышению точности

интерполяции. Для определённых кривых увеличение порядка интерполяционного полинома приводит к возрастанию погрешностей интерполяции в окрестности концов промежутка интерполяции. Этот феномен получил название *феномена Рунге*. Его проявление можно проиллюстрировать на примере *функции Рунге* $y(x) = 1/(1 + x^2)$.

График интерполяционного полинома для функции Рунге изображён на рис. 9.1.

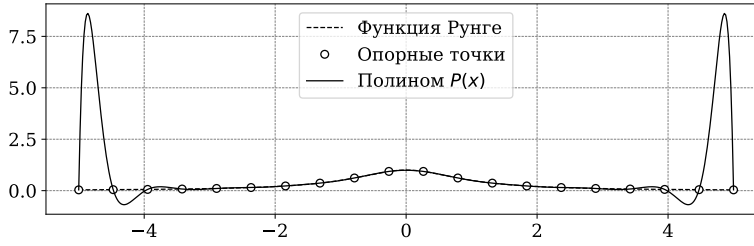

Рис. 9.1. Иллюстрация феномена Рунге на примере полинома 19-го порядка

Из графика можно видеть, что на концах промежутка интерполяционный полином претерпевает колебания, амплитуда которых будет возрастать при росте порядка полинома. При этом в центральной части области точность интерполяции растёт.

Чтобы избежать возникновения феномена Рунге, используют кусочную интерполяцию полиномами низких порядков.

9.3. Задание

1. Интерполируйте некоторую плоскую кривую полиномами высокого порядка. Визуализируйте результаты, создав анимацию изменения кривой при увеличении количества точек. Определите, выявляется ли феномен Рунге для вашего случая. Дополнительные указания по выполнению задания см. в разделе 9.4.1.
2. Для заданного набора точек $\mathbf{p}_0, \mathbf{p}_1, \ldots, \mathbf{p}_N$ некоторой двумерной кривой получите гладкую интерполяцию. Вычислите значений касательных векторов в точках по приближенным формулам и постройте интерполяционные сплайны. Сравните результат двух интерполяций. Дополнительные указания по выполнению задания см. в разделе 9.4.2.

В качестве справочной литературы рекомендуется использовать следующие источники [1—6].

9.4. Порядок выполнения лабораторной работы

9.4.1. Интерполяция полиномами высокого порядка. Феномен Рунге

Требуется задать параметрически некоторую плоскую кривую. Например, выбрана следующая кривая:

$$\mathbf{r} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} t \\ \sin t + \sin t^2 + \cos 3t \end{bmatrix}.$$

Выбранную кривую следует интерполировать полиномами высокого порядка, взяв от 3-х до N точек этой кривой.

Визуализировать результаты следует, создав анимацию изменения кривой при увеличении количества точек.

Далее следует определить, выявляется ли феномен Рунге для случая выбранной вами кривой.

На рис. 9.2 и 9.3 можно видеть проявление феномена Рунге для функции вида

$$y(x) = \frac{1}{1 + x^2}.$$

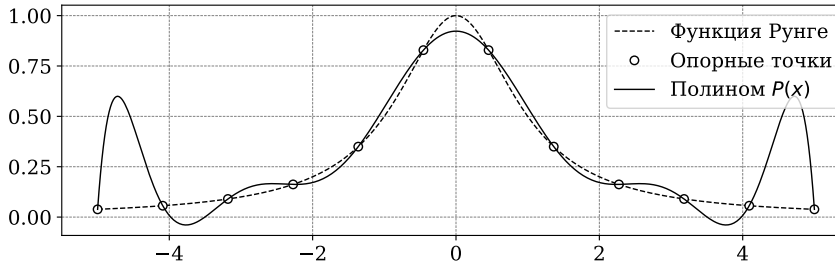

Рис. 9.2. Иллюстрация феномена Рунге для полинома порядка 11

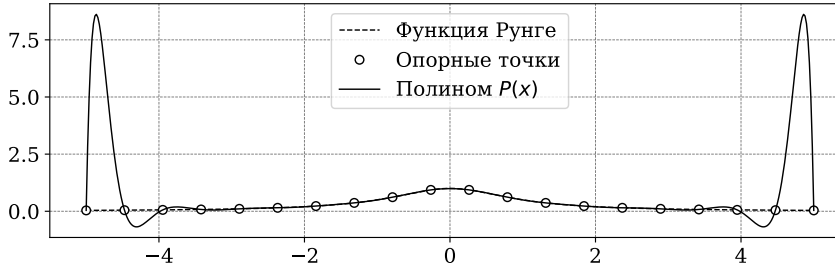

Рис. 9.3. Иллюстрация феномена Рунге для полинома порядка 19

В процессе выполнения задания необходимо решить систему линейных алгебраических уравнений. Для этого можно использовать функцию `np.linalg.solve` библиотеки `NumPy`.

9.4.2. Интерполяция функции сплайнами Эрмита

Задайте набор точек $\mathbf{p}_0$, $\mathbf{p}_1$, …, $\mathbf{p}_N$ некоторой двумерной кривой на выбор. Для некоторой двумерной кривой точки $\mathbf{p}_0$, $\mathbf{p}_1$, …, $\mathbf{p}_N$ не должны располагаться близко друг к другу, так как иначе даже интерполяция ломаной даст визуально гладкий результат.

Точки необходимо соединить кубическими сплайнами Эрмита для получения гладкой интерполяции. Следует использовать точные значения касательных векторов $\mathbf{m}_0, \mathbf{m}_1, \ldots, \mathbf{m}_N$ в точках $\{\mathbf{p}_i\}_0^N$.

Для получения значений касательных векторов в точках по приближенным формулам следует дополнить программу функциями вычисления этих значений.

Далее постройте в одной системе координат характеристическую ломаную, сплайн Эрмита и точное уравнение кривой, точки которой вы интерполировали. Маркерами выделите все опорные точки $\{\mathbf{p}_i\}_0^N$, они же точки характеристической ломаной. Например, на рис. 9.4 приведён результат интерполяции тригонометрической функции сплайном Эрмита.

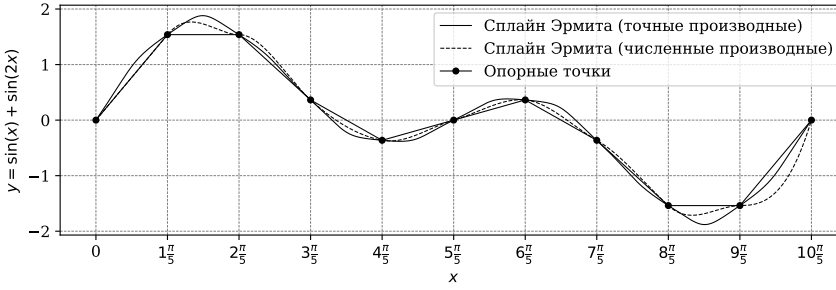

Рис. 9.4. Интерполяция сплайном Эрмита тригонометрической функции

Уменьшайте и увеличивайте количество опорных точек. При каком минимальном числе точек интерполяция выглядит визуально наиболее приемлемо? Например, из рис. 9.5 видно, что сплайн, использующий точные значения, лучше аппроксимирует окрестности экстремальных точек.

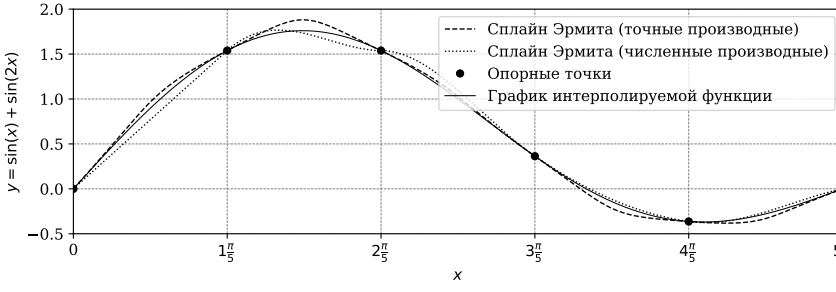

Рис. 9.5. Сравнение интерполяции с использованием точных значений касательного вектора и вычисленных по приближенным формулам

9.5. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:

- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку задания;
- описание результатов выполнения задания:
  - полученные вами графические изображения;
  - листинги (исходный код) программ (если они есть);
- выводы, согласованные с целью работы.

9.6. Контрольные вопросы

1. Что такое сплайн и чем он отличается от полинома?
2. Какие сложности могут проявиться при интерполяции полиномами высокого порядка? Как этих сложностей избежать?
3. Запишите параметрическое уравнение для ломаной линии.

Список литературы

1. *Роджерс Д.*, *Адамс А.* Математические основы машинной графики. — М. : Мир, 2001. — ISBN 5030021434.
2. *Голованов Н. Н.* Геометрическое моделирование. — М. : Физматлит, 2002. — ISBN 5940520480.
3. *Дубровин Б. А.*, *Новиков С. П.*, *Фоменко А. Т.* Современная геометрия: Методы и приложения. Т. 1. — 6-е изд. — М. : УРСС: Книжный дом «ЛИБРОКОМ», 2013. — ISBN 978-5-453-00047-0.
4. *Фиников С. П.* Курс дифференциальной геометрии. — М. : URSS, 2017.
5. Python home site. — 2018. — URL: `https://www.python.org/`.
6. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 10. Сплайн Лагранжа, кубические сплайны с хордовой и нормализованной интерполяцией

10.1. Цель лабораторной работы

Целью данной лабораторной работы является знакомство с интерполяцией с помощью сплайна Лагранжа и кубических сплайнов с хордовой и нормализованной интерполяцией.

10.2. Предварительные сведения

10.2.1. Сплайн Лагранжа

Сплайн Лагранжа для опорных точек $\{\mathbf{p}_i\}_0^N$ строится по следующей формуле:

$$\mathbf{r}(t) = \sum_{i=0}^{N} L_i(t)\mathbf{p}_i, \quad L_i(t) = \prod_{j=0, j\neq i}^{N} \frac{t - t_j}{t_i - t_j}, \quad t \in [t_0, t_N].$$

Никаких дополнительных требований не накладывается, и для построения необходимо задать лишь набор опорных точек.

Функции $L_i(t)$ являются базисными функциями для полинома Лагранжа. Вышеизложенная формула подразумевает, что построение производится по всему массиву опорных точек сразу. Из-за этого с увеличением числа точек растёт порядок полинома, что может привести к возникновению эффекта Рунге на концах интерполируемого отрезка.

Сплайн Лагранжа, таким образом, является одним из самых простых сплайнов и подходит для применения лишь в несложных и учебных задачах.

10.2.2. Кубический сплайн

Кубическим сплайном принято называть параметрический сплайн, который в точках соединения своих сегментов имеет непрерывные первые и вторые производные. Уравнение одного сегмента такого сплайна имеет вид

$$\mathbf{r}_i(t) = \mathbf{a}_0 + \mathbf{a}_1 t + \mathbf{a}_2 t^2 + \mathbf{a}_3 t^3, \quad t_i \leqslant t \leqslant t_{i+1}, \quad i = 0, \ldots, N.$$

Без потери общности можно положить $0 \leqslant t \leqslant t_{i+1}$. В этом случае говорят о *хордовой* интерполяции с помощью кубических сплайнов.

Встаёт задача нахождения коэффициентов сплайна. Для каждого сегмента сплайна необходимо задать две опорные точки и два касательных вектора. Для примера рассмотрим первый сегмент сплайна.

Пусть заданы точки $\mathbf{p}_0, \mathbf{p}_1$ и касательные векторы $\dot{\mathbf{p}}_0, \dot{\mathbf{p}}_1$. Решая систему уравнений

$$\mathbf{r}(0) = \mathbf{p}_0, \ \mathbf{r}(t_1) = \mathbf{p}_1, \ \dot{\mathbf{r}}(0) = \dot{\mathbf{p}}_0, \ \dot{\mathbf{r}}(t_1) = \dot{\mathbf{p}}_1,$$

относительно коэффициентов $\mathbf{a}_0$, $\mathbf{a}_1$, $\mathbf{a}_2$, $\mathbf{a}_3$, получим следующие выражения для коэффициентов:

$$\mathbf{a}_0 = \mathbf{p}_0, \ \mathbf{a}_0 = \dot{\mathbf{p}}_0, \ \mathbf{a}_2 = \frac{3(\mathbf{p}_1 - \mathbf{p}_0)}{t_1^2} - \frac{2\dot{\mathbf{p}}_0}{t_1} - \frac{\dot{\mathbf{p}}_1}{t_1}, \ \mathbf{a}_3 = \frac{2(\mathbf{p}_0 - \mathbf{p}_1)}{t_1^3} + \frac{\dot{\mathbf{p}}_0}{t_1^2} + \frac{\dot{\mathbf{p}}_1}{t_1^2}.$$

Аналогичные выражения получаются и для всех остальных сегментов сплайна в точках $\mathbf{p}_2$, $\mathbf{p}_3$, $\mathbf{p}_4$ и т.д. до $\mathbf{p}_N$. Параметр t на каждом отрезке $[0, t_{i+1}]$ изменяется от 0 до t_{i+1}. Значение t_{i+1} при хордовой интерполяции вычисляют как длину между соседними опорными точками:

$$t_{i+1} = \|\mathbf{p}_{i+1} - \mathbf{p}_i\|.$$

В задачах, чаще всего встречающихся на практике, значения производных $\dot{\mathbf{p}}_i$ во всех точках не известны. Однако, воспользовавшись условием непрерывности вторых производных в каждой точке кубического сплайна, можно получить систему линейных уравнений, которая позволит вычислить касательные векторы во всех опорных точках кроме начальной $\mathbf{p}_0$ и конечной $\mathbf{p}_N$. Подробные выкладки можно найти в книге [1]. Здесь же мы приведём лишь основные формулы.

Значения касательной находятся из следующей системы:

$$\mathbf{M} \cdot \dot{\mathbf{P}} = \mathbf{R},$$

где матрица $\mathbf{M}$ имеет размерность $(N+1) \times (N+1)$ и является трёхдиагональной:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ t_2 & 2(t_1+t_2) & t_1 & \dots & 0 & 0 & 0 & 0 \\ 0 & t_3 & 2(t_2+t_3) & \dots & 0 & 0 & 0 & 0 \\ 0 & 0 & t_4 & \dots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & t_{N-1} & 2(t_{N-1}+t_{N-2}) & t_{N-2} & 0 \\ 0 & 0 & 0 & \dots & 0 & t_N & 2(t_N+t_{N-1}) & t_{N-1} \\ 0 & 0 & 0 & \dots & 0 & 0 & 0 & 1 \end{pmatrix}$$

Матрица $\dot{\mathbf{P}}$ имеет размерность $(N+1)\times 3$ (для плоских кривых размерность матрицы $(N+1)\times 2$) и состоит из компонент касательных векторов, относительно которых и необходимо решать данную линейную систему алгебраических уравнений:

$$\dot{\mathbf{P}} = \begin{pmatrix} \dot{x}_0 & \dot{y}_0 & \dot{z}_0 \\ \dot{x}_1 & \dot{y}_1 & \dot{z}_1 \\ \vdots & \vdots & \vdots \\ \dot{x}_N & \dot{y}_N & \dot{z}_N \end{pmatrix}.$$

Наконец вектор $\mathbf{R}$ имеет следующий вид:

$$\mathbf{R} = \begin{pmatrix} \dot{\mathbf{p}}_0 \\ \frac{3}{t_1 t_2}(t_1^2(\mathbf{p}_2 - \mathbf{p}_1) + t_2^2(\mathbf{p}_1 - \mathbf{p}_0)) \\ \frac{3}{t_2 t_3}(t_2^2(\mathbf{p}_3 - \mathbf{p}_2) + t_3^2(\mathbf{p}_2 - \mathbf{p}_1)) \\ \vdots \\ \frac{3}{t_{N-1} t_N}(t_{N-1}^2(\mathbf{p}_N - \mathbf{p}_{N-1}) + t_N^2(\mathbf{p}_{N-1} - \mathbf{p}_{N-2}))\mathbf{p}_N \\ \dot{\mathbf{p}}_N \end{pmatrix}.$$

Касательные векторы $\dot{\mathbf{p}}_0$ и $\dot{\mathbf{p}}_N$ предполагаются известными.

После нахождения касательных векторов можно вычислить коэффициенты для каждого сегмента кубического сплайна:

$$\mathbf{r}(t) = \begin{pmatrix} F_{1i}(\tau) & F_{2i}(\tau) & F_{3i}(\tau) & F_{4i}(\tau) \end{pmatrix} \begin{pmatrix} \mathbf{P}_i \\ \mathbf{P}_{i+1} \\ \dot{\mathbf{P}}_i \\ \dot{\mathbf{P}}_{i+1} \end{pmatrix},$$

где $\tau = t/t_{i+1}$, а функции F называются *весовыми* функциями и имеют следующий вид:

$$F_{1i}(\tau) = 2\tau^3 - 3\tau^2 + 1, \quad F_{2i}(\tau) = -2\tau^3 + 3\tau^2,$$
$$F_{3i}(\tau) = \tau(\tau^2 - 2\tau + 1)t_{i+1}, \quad F_{4i}(\tau) = \tau(\tau^2 - \tau)t_{i+1}.$$

Таким образом, вычисление кубического сплайна включает в себя следующие шаги:
- по заданным опорным точкам вычисляются значения параметра t_i;
- решается система линейных алгебраических уравнений, из которой находятся значения касательных в опорных точках;
- по опорным точкам и касательным вычисляются коэффициенты каждого сегмента сплайна;
- путём изменения значений параметра t производится интерполяция искомой кривой.

Кубические сплайны оказались особенно полезны для численных методов решения обыкновенных дифференциальных уравнений с плотной выдачей, так как в этом случае значения производных известны в каждой точке из правой части дифференциального уравнения.

10.2.3. Сплайн Эрмита

Сплайны Эрмита названы в честь французского математика Шарля Эрмита (Charles Hermite, 1822–1901) и являются кубическими полиномами, проходящими через две точки $\mathbf{p}_i$ и $\mathbf{p}_{i+1}$ при известных производных в этих точках $\mathbf{m}_i$ и $\mathbf{m}_{i+1}$. В компьютерной графике данные сплайны часто носят название `cspline`.

Для простоты обозначений рассмотрим две точки $\mathbf{p}_0$, $\mathbf{p}_1$ и два касательных вектора $\mathbf{m}_0$, $\mathbf{m}_1$. Формула для вычисления точек сплайна имеет следующий вид:

$$\mathbf{r}(t) = h_{00}(t)\mathbf{p}_0 + h_{10}(t)\mathbf{m}_0 + h_{01}(t)\mathbf{p}_1 + h_{11}(t)\mathbf{m}_1, \ t \in [0, 1].$$

Функции $h_{00}(t)$, $h_{01}(t)$, $h_{10}(t)$ и $h_{11}(t)$ являются базисными полиномами. Их можно вычислить, воспользовавшись следующими граничными условиями:

$$\mathbf{r}(0) = \mathbf{p}_0, \ \mathbf{r}(1) = \mathbf{p}_1, \ \mathbf{r}'(0) = \mathbf{m}_0, \ \mathbf{r}'(1) = \mathbf{m}_1.$$

На практике используют три формы записи данных функций:
- **Приведённая форма** совпадает с нормализованными весовыми функциями для кубических сплайнов:

$$h_{00}(t) = 2t^3 - 3t^2 + 1, \quad h_{10}(t) = t^3 - 2t^2 + t,$$
$$h_{01}(t) = -2t^3 + 3t^2, \qquad h_{11}(t) = t^3 - t^2.$$

– **Факторизированная форма** позволяет минимизировать число арифметических операций при вычислении:

$$h_{00}(t) = (1+2t)(1-t)^2, \quad h_{10}(t) = t(1-t)^2,$$
$$h_{01}(t) = t^2(3+2t), \qquad h_{11}(t) = t^2(t-1).$$

– **Форма Безье** показывает связь между сплайнами Эрмита и полиномами Бернштейна (см. далее):

$$h_{00}(t) = B_3^0(t) + B_3^1(t), \qquad h_{10}(t) = B_3^1(t)/3,$$
$$h_{01}(t) = B_3^3(t) + B_3^2(t), \quad h_{11}(t) = -B_3^2(t)/3.$$

Факторизированная форма помогает сразу же выяснить, что $h_{00}(1) = h_{10}(1) = 0$, $h_{11}(1) = 0$, $h_{01}(1) = 1$ и $h_{01}(0) = h_{10}(0) = 0$, $h_{11}(0) = 0$, $h_{00}(0) = 1$.

Третья форма позволяет выявить связь сплайнов Эрмита с кривыми Безье и свести их построение к построению кривых Безье. Для этого можно вычислить опорные точки кривой Безье по следующим формулам:

$$\mathbf{w}_0 = \mathbf{p}_0,\ \mathbf{w}_1 = \mathbf{p}_0 + \mathbf{m}_0/3,\ \mathbf{w}_2 = \mathbf{p}_1 - \mathbf{m}_1/3,\ \mathbf{w}_3 = \mathbf{p}_1.$$

Сплайны Эрмита можно использовать для аппроксимации данных без известных значений производных (касательных). Для этого необходимо вычислить значения производных в точках по приближенным формулам.

Пусть дан набор точек $\{(t_k, \mathbf{p}_k)\}_1^n$, который необходимо соединить гладкой кривой. При использовании сплайнов Эрмита интерполяционная кривая будет состоять из отдельных сплайнов Эрмита, соединяющих пары точек. Кривая будет гладко дифференцируемой на всём промежутке $[t_0, t_n]$.

Приближенное вычисление касательных векторов (производных) можно проводить разными способами. Перечислим здесь несколько формул.

Простейший способ заключается в использовании трёхточечных конечных разностей:

$$\mathbf{m}_k = \frac{1}{2}\left(\frac{\mathbf{p}_{k+1} - \mathbf{p}_k}{t_{k+1} - t_k} + \frac{\mathbf{p}_k - \mathbf{p}_{k-1}}{t_k - t_{k+1}}\right),\ k = 2, \dots, n-1.$$

В начальной и конечной точке придётся использовать обычные конечные разности:

$$\mathbf{m}_0 = \frac{\mathbf{p}_1 - \mathbf{p}_0}{t_1 - t_0},\ \mathbf{m}_n = \frac{\mathbf{p}_n - \mathbf{p}_{n-1}}{t_n - t_{n-1}}.$$

Другим способом является использование формулы

$$\mathbf{m}_k = (1-c)\frac{\mathbf{p}_{k+1} - \mathbf{p}_{k-1}}{t_{k+1} - t_{k-1}},\ c \in [0,1].$$

При $c = 0$ сплайн Эрмита сведётся к сплайну Катмулл–Рома.

Ещё одна формула имеет вид

$$\mathbf{m}_k = s_{k+1}\frac{\mathbf{p}_k - \mathbf{p}_{k-1}}{s_k + s_{k+1}} + s_k\frac{\mathbf{p}_{k+1} - \mathbf{p}_k}{s_k + s_{k+1}},\ s_k = |\mathbf{p}_k - \mathbf{p}_{k-1}|,\ s_{k+1} = |\mathbf{p}_{k+1} - \mathbf{p}_k|.$$

В конечных точках найдём $\mathbf{m}_0$ и $\mathbf{m}_n$ из условия равенства нулю третьей производной радиус-вектора $\mathbf{r}'''(t) = 0$.

10.3. Задание

1. Задайте набор опорных точек $\mathbf{p}_0$, $\mathbf{p}_1$, …, $\mathbf{p}_N$ произвольной двумерной кривой. Для проверки результатов используйте следующий набор точек: $\mathbf{p}_0 = (0.0, 0.0)^T$, $\mathbf{p}_1 = (1.0, 1.0)^T$, $\mathbf{p}_2 = (2.0, -1.0)^T$, $\mathbf{p}_3 = (3.0, 0.0)^T$. Данный набор точек взят из примера 5-2 книги [1].
2. Постройте точки ломаной линии по заданным опорным точкам так, как это изображено на рис. 10.1.

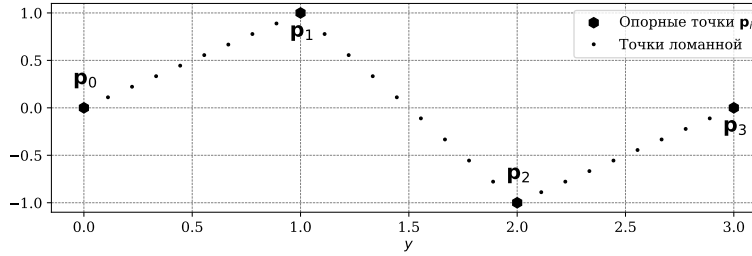

Рис. 10.1. Точки ломаной для опорных точек $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$

3. Постройте сплайн Лагранжа по заданным опорным точкам. Изобразите на одном рисунке построенный сплайн, ломаную линию и опорные точки (см. рис. 10.2).

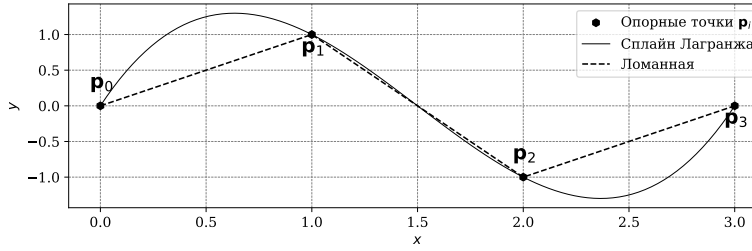

Рис. 10.2. Сплайн Лагранжа для опорных точек $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$

Обратите внимание, что использовать для интерполяции необходимо именно **сплайны**, а не одноимённый полином Лагранжа.

4. Постройте кубический сплайн, используя лишь заданные опорные точки. Так как для построения кубического сплайна необходимо задать производные $\dot{\mathbf{p}}_0$, $\dot{\mathbf{p}}_N$ в начальной и конечной точках, а по условию производные неизвестны, то вычислите их приближённо для хордовой интерполяции по формулам:

$$\dot{\mathbf{p}}_0 \approx \frac{\mathbf{p}_1 - \mathbf{p}_0}{\|\mathbf{p}_1 - \mathbf{p}_0\|}, \quad \dot{\mathbf{p}}_N \approx \frac{\mathbf{p}_N - \mathbf{p}_{N-1}}{\|\mathbf{p}_N - \mathbf{p}_{N-1}\|}.$$

На рис. 10.2 приведены два кубических сплайна, один из которых построен с помощью хордовой аппроксимации, а второй — с помощью нормализованной аппроксимации.

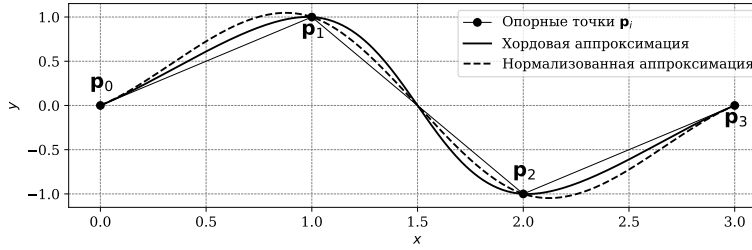

Рис. 10.3. Кубический сплайн для опорных точек $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$

5. Определите, какой из сплайнов лучше аппроксимировал выбранную вами кривую.
6. Изобразить кривую, точки и все сплайны в одной системе координат. Видна ли разница в интерполяции «на глаз»? Проявляется ли эффект Рунге в случае сплайна Лагранжа?

В качестве справочной литературы рекомендуется использовать следующие источники [1—6].

10.4. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:
- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку задания;
- описание результатов выполнения задания:
  - полученные вами графические изображения;
  - листинги (исходный код) программ (если они есть);
- выводы, согласованные с целью работы.

10.5. Контрольные вопросы

1. Запишите формулы для кубического сплайна. Чем кубический сплайн отличается от кубического полинома?
2. Запишите формулы для сплайна Лагранжа. Чем сплайн отличается от одноимённого полинома?
3. Запишите формулы для сплайна Эрмита. Какие формы записи его коэффициентов вы знаете?
4. Каковы преимущества и недостатки сплайнов Эрмита по сравнению со сплайном Лагранжа и кубическими сплайнами?
5. Чем сплайн Эрмита отличается от одноименных полиномов?

Список литературы

1. *Роджерс Д.*, *Адамс А.* Математические основы машинной графики. — М. : Мир, 2001. — ISBN 5030021434.
2. *Голованов Н. Н.* Геометрическое моделирование. — М. : Физматлит, 2002. — ISBN 5940520480.
3. *Дубровин Б. А.*, *Новиков С. П.*, *Фоменко А. Т.* Современная геометрия: Методы и приложения. Т. 1. — 6-е изд. — М. : УРСС: Книжный дом «ЛИБРОКОМ», 2013. — ISBN 978-5-453-00047-0.
4. *Фиников С. П.* Курс дифференциальной геометрии. — М. : URSS, 2017.
5. Python home site. — 2018. — URL: `https://www.python.org/`.
6. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 11. Кривые Безье

11.1. Цель лабораторной работы

Целью данной лабораторной работы является изучение свойств кривых Безье.

11.2. Предварительные сведения

Кривые Безье названы в честь французского инженера компании Рено (Renault) Пьера Безье (Pierre Bézier, 1910–1999). Он опубликовал результаты своих исследований в 1962 году. Однако он был не первым, кто использовал данные кривые. К похожим результатам пришёл другой французский инженер компании Ситроен (Citroën) — Поль де Кастельё (Paul de Casteljau, 1930) в 1959 году, но свои результаты он не опубликовывал. Кроме того, кривые Безье по своей сути являются полиномами Бернштейна, названными так в честь русского математика Сергея Натановича Бернштейна (1880–1968).

Кривые Безье нашли своё применение во многих областях инженерного дела, компьютерной графики и дизайна. Они активно используются в следующих типах программ:

- в системах автоматизированного проектирования (CAD и CAM), которые активно используются в инженерии, архитектуре и в машиностроении;
- в векторных графических редакторах, таких как Adobe Illustrator, CorelDRAW, Inkscape;
- в векторных форматах графических изображений, таких как SVG, OpenType fonts, PDF, PS, и многих других;
- в специализированных языках программирования для построения векторных изображений: Tikz/PGF, Asymptote.

11.2.1. Основные определения

Базисом Бернштейна называется совокупность функций следующего вида:

$$B_n^i(t) = \mathrm{C}_n^i (1-t)^{n-i} t^i, \quad t \in [0,1],$$

где $i, n = 0, 1, 2, 3, \ldots$ — индексы, а параметр t принимает значения из единичного промежутка.

Кривая Безье является полиномом Бернштейна с векторными коэффициентами $\mathbf{w}_i$:

$$\mathbf{B}_n(t) = \sum_{i=0}^{n} B_n^i(t)\mathbf{w}_i,$$

где $\mathbf{w}_i = (x_i, y_i, z_i)$ в трёхмерном случае и $\mathbf{w}_i = (x_i, y_i)$ в двумерном случае.

Кривая проходит через начальную $\mathbf{w}_0$ и конечную $\mathbf{w}_n$ точки. Точки $\{\mathbf{w}_0, \mathbf{w}_1, \ldots, \mathbf{w}_n\}$ формирует опорный «скелет» (или «каркас») и называются *управляющими* (или *контрольными*) точками кривой Безье. Ломаная, образующаяся при последовательном соединении этих точек, называется *характеристической ломаной* кривой Безье.

Часто управляющими точками называют лишь точки $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{n-1}$, лежащие вне кривой, отличая их таким образом от конечных точек $\mathbf{w}_0$ и $\mathbf{w}_n$, через которые проходит кривая.

Вычислим несколько первых функций Бернштейна (рис. 11.1):

$$B_0^0 = 1, \quad B_1^0 = 1 - t, \quad B_1^1 = t.$$

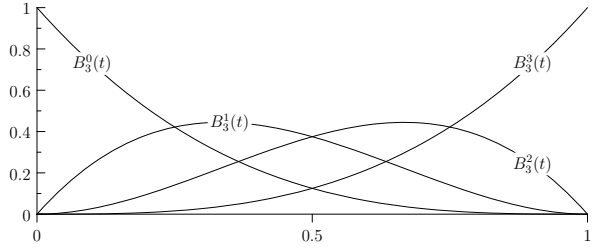

Рис. 11.1. Графики нескольких функций Бернштейна

Для вычисления других функций можно воспользоваться рекуррентной формулой

$$B_n^i = tB_{n-1}^{i-1} + (1-t)B_{n-1}^i,$$

которая доказывается путём подстановки выражений для B_{n-1}^{i-1} и B_{n-1}^i.

Базис Бернштейна представляет собой разложение единицы. Это можно показать, воспользовавшись формулой для бинома Ньютона:

$$\sum_{i=0}^{n} B_n^i(t) = (t + (1-t))^n = 1^n = 1.$$

На практике используют кривые Безье 2-го, 3-го и 4-го порядков. При реализации кривых более высокого порядка самым ресурсоёмким шагом является вычисление биноминальных коэффициентов C_n^i, поэтому разумным является вычисление этих коэффициентов заранее и сохранение их во вложенных списках.

На рис. 11.2 изображены три кривые Безье второго порядка с одинаковыми начальной и конечной точками $\mathbf{w}_0$ и $\mathbf{w}_2$, но с разной контрольной точкой $\mathbf{w}_1$. Одной контрольной точки часто бывает недостаточно. Тогда используют кривые Безье третьего порядка (см. рис. 11.3), для построения которых нужны уже четыре опорные точки, две из которых являются контрольными.

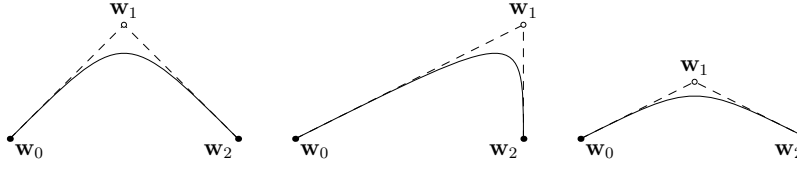

Рис. 11.2. Кривые Безье второго порядка. На рисунке показано различное положение контрольной точки $\mathbf{w}_1$

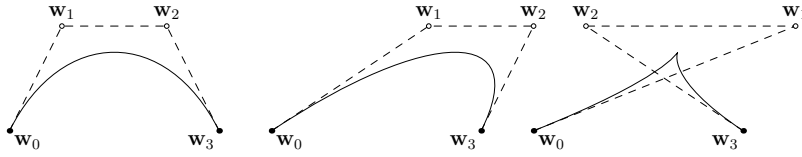

Рис. 11.3. Кривые Безье третьего порядка и их характеристические ломаные. Две контрольные точки $\mathbf{w}_1$ и $\mathbf{w}_2$ позволяют более гибко управлять положением кривой

В графических векторных редакторах манипуляция кривыми Безье осуществляется путём перемещения опорных точек с помощью мыши. Вначале указываются желаемые конечные точки, а затем с помощью контрольных точек кривая подстраивается под нужную форму.

Выше упоминалось, что кривые Безье используются в шрифтах форматов True Type и Open Type. Каждый глиф шрифта состоит из одного или нескольких контуров, заданных с помощью кривых Безье третьего порядка с четырьмя опорными точками, две из которых управляющие. На рис. 11.4 представлено несколько примеров.

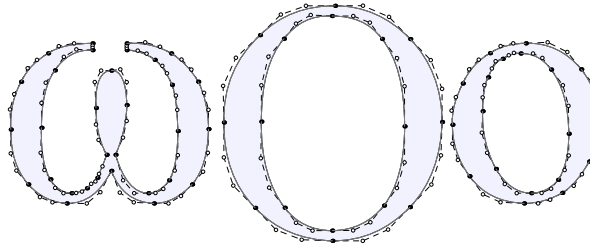

Рис. 11.4. Контуры букв ω, O и o шрифта Times New Roman. Показаны опорные и управляющие точки кривых Безье

11.2.2. Матричные формулы для вычисления кривых Безье

Для ускорения вычислений точек кривой Безье можно воспользоваться матричной формулировкой вышеприведённых формул. Современные процессоры хорошо поддерживают операции с векторами, поэтому выигрыш в производительности можно получить даже при выполнении кода на одноядерном процессоре. В случае использования библиотеки `NumPy` ускорение достигается как за счёт вызовов быстрых функций на Си, так и за счёт распараллеливания операций с массивами.

Рассмотрим случай кубической и квадратной кривых Безье. При необходимости можно аналогичным способом получить формулы и для кривых более высокого порядка.

Кубическая кривая Безье имеет следующий вид:

$$\mathbf{B}_3(t) = (1-t)^3\mathbf{w}_0 + 3(1-t)^2 t\mathbf{w}_1 + 3(1-t)t^2\mathbf{w}_2 + t^3\mathbf{w}_3.$$

Раскроем скобки в каждом слагаемом, однако подобные приводить не будем. Также временно уберём веса, заменив их на единицу, и запишем слагаемые одно под другим столбиком:

$$B_3(t) = \begin{pmatrix} 1 + 3t^2 - 3t - t^3 + \\ +3(t - 2t^2 + t^3) + \\ +3(t^2 - t^3) + \\ +t^3 \end{pmatrix} = \begin{pmatrix} -1t^3 & +3t^2 & -3t & +1 & + \\ +3t^3 & -6t^2 & +3t & +0 & + \\ -3t^3 & +3t^2 & +0t & +0 & + \\ +1t^3 & +0t^2 & +0t & +0 & \end{pmatrix}.$$

Разложим теперь данную сумму на произведения вектора-строки $(t^3, t^2, t^1, 1)$ и векторов-столбцов, составленных из коэффициентов:

$$\mathbf{B}_3(t) = \begin{pmatrix} t^3 & t^2 & t^1 & 1 \end{pmatrix} \begin{pmatrix} -1 \\ 3 \\ -3 \\ 1 \end{pmatrix} + \begin{pmatrix} t^3 & t^2 & t^1 & 1 \end{pmatrix} \begin{pmatrix} -3 \\ -6 \\ 3 \\ 0 \end{pmatrix} +$$

$$+ \begin{pmatrix} t^3 & t^2 & t^1 & 1 \end{pmatrix} \begin{pmatrix} -3 \\ -3 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} t^3 & t^2 & t^1 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix},$$

$$\mathbf{B}_3(t) = \begin{pmatrix} t^3 & t^2 & t^1 & 1 \end{pmatrix} \begin{pmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & t^1 & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}.$$

Вернём опорные точки:

$$\mathbf{B}_3(t) = \begin{pmatrix} 1 & t^1 & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} \mathbf{w}_0 \\ \mathbf{w}_1 \\ \mathbf{w}_2 \\ \mathbf{w}_3 \end{pmatrix} =$$

$$= \begin{pmatrix} 1 & t^1 & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{pmatrix}.$$

В результате получим

$$\mathbf{B}_3(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}.$$

Для кривой второго порядка аналогично можно вывести

$$\mathbf{B}_2(t) = \begin{pmatrix} 1 & t^1 & t^2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ -2 & 2 & 0 \\ 1 & -2 & 1 \end{pmatrix} \begin{pmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \end{pmatrix}.$$

11.2.3. Алгоритм де Кастельё

Можно сформулировать итерационную процедуру, которая позволит вычислять точки кривой Безье без использования функций Бернштейна. Геометрический смысл данной процедуры заключается в построении дополнительного каркаса из ломаных линий.

Выше было показано, что функции Бернштейна удовлетворяют следующему рекуррентному соотношению:

$$B_n^i(t) = t B_{n-1}^{i-1}(t) + (1-t) B_{n-1}^i(t).$$

Используя это соотношение, можно преобразовать формулу для кривых Безье:

$$\mathbf{r}(t) = \sum_{i=0}^{n} B_n^i(t)\mathbf{p}_i = B_n^0(t)\mathbf{p}_0 + B_n^n(t)\mathbf{p}_n + \sum_{i=1}^{n-1} B_n^i(t)\mathbf{p}_i.$$

Учитывая, что $B_n^0(t) = (1-t)^n$ и $B_n^n(t) = t^n$, получим

$$\begin{aligned} \mathbf{r}(t) &= (1-t)^n\mathbf{p}_0 + t^n\mathbf{p}_n + \sum_{i=1}^{n-1}(t B_{n-1}^{i-1}(t) + (1-t)B_{n-1}^i(t))\mathbf{p}_i = \\ &= (1-t)^n\mathbf{p}_0 + \sum_{i=1}^{n-1} B_{n-1}^i(t)\mathbf{p}_i + t^n\mathbf{p}_n + \sum_{i=1}^{n-1} t B_{n-1}^{i-1}(t)\mathbf{p}_i = \\ &= (1-t)\left[(1-t)^{n-1}\mathbf{p}_0 + \sum_{i=1}^{n-1} B_{n-1}^i(t)\mathbf{p}_i\right] + t\left[t^{n-1}\mathbf{p}_n + \sum_{i=1}^{n-1} B_{n-1}^{i-1}(t)\mathbf{p}_i\right]. \end{aligned}$$

Для дальнейшего преобразования заметим, что $B_{n-1}^0(t) = (1-t)^{n-1}$, из чего следует, что $(1-t)^{n-1}\mathbf{p}_0 = B_{n-1}^0(t)\mathbf{p}_0$ и $B_{n-1}^{n-1}(t) = t^{n-1}$. Из этого, в свою очередь, следует, что $t^{n-1}\mathbf{p}_n = B_{n-1}^0\mathbf{p}_n$. Используя эти соотношения, продолжим преобразовывать формулу для кривой Безье:

$$(1-t)\left[(1-t)^{n-1}\mathbf{p}_0 + \sum_{i=1}^{n-1} B_{n-1}^i(t)\mathbf{p}_i\right] + t\left[t^{n-1}\mathbf{p}_n + \sum_{i=1}^{n-1} B_{n-1}^{i-1}(t)\mathbf{p}_i\right] =$$

$$= (1-t)\left(\sum_{i=0}^{n-1} B_{n-1}^{i}(t)\mathbf{p}_i\right) + t\left(\sum_{i=0}^{n-1} B_{n-1}^{i}\mathbf{p}_{i+1}\right) = (1-t)\mathbf{r}_{n-1}^{0}(t) + t\mathbf{r}_{n-1}^{1}(t).$$

Мы ввели следующие обозначения:

$$\mathbf{r}_{n-1}^{0}(t) = \sum_{i=0}^{n-1} B_{n-1}^{i}(t)\mathbf{p}_i, \quad \mathbf{r}_{n-1}^{1} = \sum_{i=0}^{n-1} B_{n-1}^{i}(t)\mathbf{p}_{i+1}.$$

Продолжая процесс разложения дальше, приходим к равенствам:

$$\mathbf{r}_0^i = \sum_{j=0}^{0} B_0^0(t)\mathbf{p}_i = \mathbf{p}_i, \quad i = 0, 1, \ldots, n, \quad \mathbf{r}_n^0(t) = \mathbf{r}(t), \quad \mathbf{r}_0^i = \mathbf{p}_i.$$

Таким образом, получена рекуррентная формула, которая позволяет в результате итерационного процесса вычислить координаты точек кривой Безье, используя в качестве начальных данных лишь опорные точки $\mathbf{p}_i$. При это пропадает необходимость вычислять функции Бернштейна:

$$\mathbf{r}_k^i = (1-t)\mathbf{r}_{k-1}^i + t\mathbf{r}_{k-1}^{i+1}, \quad i + k \leqslant n.$$

Данная формула называется *алгоритмом де Кастельё*. Геометрический смысл этого алгоритма заключается в вычислении вспомогательных характеристических ломаных, которые строятся друг за другом (рис. 11.5).

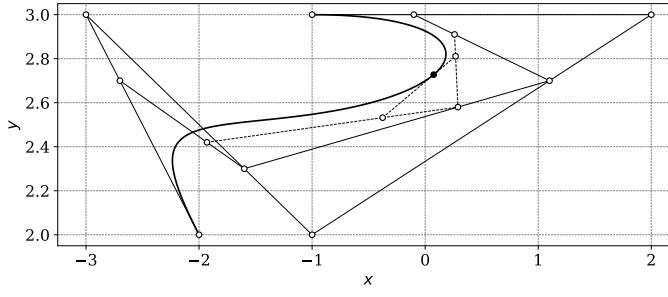

Рис. 11.5. Вспомогательные характеристические (опорные) ломаные, точки которых вычисляет алгоритм де Кастельё

Чтобы лучше понять этот алгоритм, рассмотрим вычисление для кривой Безье 3-го порядка. Начинаем с 4-х опорных точек $\mathbf{p}_0, \mathbf{p}_1, \mathbf{p}_2, \mathbf{p}_3$ и с их помощью вычисляем три новые опорные точки:

$$\mathbf{r}_1^0 = (1-t)\mathbf{r}_0^0 + t\mathbf{r}_0^1 = (1-t)\mathbf{p}_0 + t\mathbf{p}_1,$$
$$\mathbf{r}_1^1 = (1-t)\mathbf{r}_0^1 + t\mathbf{r}_0^2 = (1-t)\mathbf{p}_1 + t\mathbf{p}_2,$$
$$\mathbf{r}_1^2 = (1-t)\mathbf{r}_0^2 + t\mathbf{r}_0^3 = (1-t)\mathbf{p}_2 + t\mathbf{p}_3.$$

Вычисленные три точки $\mathbf{r}_1^0$, $\mathbf{r}_1^1$, $\mathbf{r}_1^2$ используем для вычисления следующих двух:

$$\mathbf{r}_2^0 = (1-t)\mathbf{r}_1^0 + t\mathbf{r}_1^1, \quad \mathbf{r}_2^1 = (1-t)\mathbf{r}_1^1 + t\mathbf{r}_2^1.$$

Наконец, последние две точки $\mathbf{r}_2^0$, $\mathbf{r}_2^1$ дают нам возможность вычислить точку непосредственно на самой кривой Безье:

$$\mathbf{r}_3^0 = (1-t)\mathbf{r}_2^0 + t\mathbf{r}_2^1.$$

11.2.4. Нахождение производных от кривых Безье

Одним из полезных свойств кривой Безье является возможность вычислить значение производной в произвольной точке, лишь пересчитав опорные точки. Пусть заданы опорные точки $\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_n$ и по ним построена кривая Безье:

$$\mathbf{r}(t) = \sum_{i=0}^{n} B_n^i(t)\mathbf{w}_i.$$

Можно доказать следующую формулу:

$$\frac{r(t)}{t} = \sum_{i=0}^{n-1} B_{n-1}^i(t) n(\mathbf{w}_{i+1} - \mathbf{w}_i),$$

где $n(\mathbf{w}_{i+1} - \mathbf{w}_i)$ — новый набор опорных точек для кривой $(n-1)$-го порядка, которой и является производная от исходной кривой Безье. Для доказательства этой формулы необходимо вычислить производные от функций Бернштейна и перегруппировать их сумму.

Приведём простейший пример для кривой третьего порядка. Пусть даны четыре опорные точки $\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{w}_2$, $\mathbf{w}_3$. Тогда опорные точки, соответствующие первой производной, будут иметь вид

$$\{\mathbf{w}_0', \mathbf{w}_1', \mathbf{w}_2'\} = \{3(\mathbf{w}_1 - \mathbf{w}_0), 3(\mathbf{w}_2 - \mathbf{w}_1), 3(\mathbf{w}_3 - \mathbf{w}_2)\}.$$

Опорные точки для второй приводной можно вычислить аналогично:

$$\{\mathbf{w}_0'', \mathbf{w}_1''\} = \{2(\mathbf{w}_1' - \mathbf{w}_0'), 2(\mathbf{w}_2' - \mathbf{w}_1')\}.$$

Третья производная представляет собой только одну точку.

Возможность быстро вычислять производные всех возможных порядков позволяет эффективно находить касательные и нормальные векторы к кривой, а также находить экстремальные точки. Рассмотрим формулы для вычисления касательных и нормальных векторов.

Компоненты ненормированного касательного вектора находятся сразу же после вычисления производной, так как

$$\mathbf{v}(t) = \frac{r(t)}{t} = \left(\frac{x(t)}{t}, \frac{(t)}{t}\right)^T, \quad |\mathbf{v}(t)| = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)}.$$

Нормируем вектор

$$\mathbf{v}_0(t) = \frac{\mathbf{v}(t)}{|\mathbf{v}|} = \left(\frac{\dot{x}(t)}{|\mathbf{v}(t)|}, \frac{\dot{y}(t)}{|\mathbf{v}(t)|} \right)^T = \left(v_0^1, v_0^2 \right)^T.$$

Задачу нахождения нормали можно существенно упростить, если воспользоваться тем фактом, что нормаль и касательный вектор ортогональны друг другу, так что нормаль можно получить из касательного вектора путём поворота последнего на 90° (рис. 11.6):

$$\mathbf{n}_0 = \begin{pmatrix} v_0^1 \cos \pi/2 - v_0^2 \sin \pi/2 \\ v_0^1 \sin \pi/2 + v_0^2 \cos \pi/2 \end{pmatrix} = \begin{pmatrix} -v_0^2(t) \\ +v_0^1(t) \end{pmatrix} = \begin{pmatrix} \dfrac{-\dot{y}}{\sqrt{\dot{x} + \dot{y}}} \\ \dfrac{+\dot{x}}{\sqrt{\dot{x} + \dot{y}}} \end{pmatrix}.$$

Таким образом:

$$\mathbf{v}_0(t) = \begin{pmatrix} \dfrac{\dot{x}}{\sqrt{\dot{x} + \dot{y}}} \\ \dfrac{\dot{y}}{\sqrt{\dot{x} + \dot{y}}} \end{pmatrix}, \quad \mathbf{n}_0(t) = \begin{pmatrix} \dfrac{-\dot{y}}{\sqrt{\dot{x} + \dot{y}}} \\ \dfrac{\dot{x}}{\sqrt{\dot{x} + \dot{y}}} \end{pmatrix}.$$

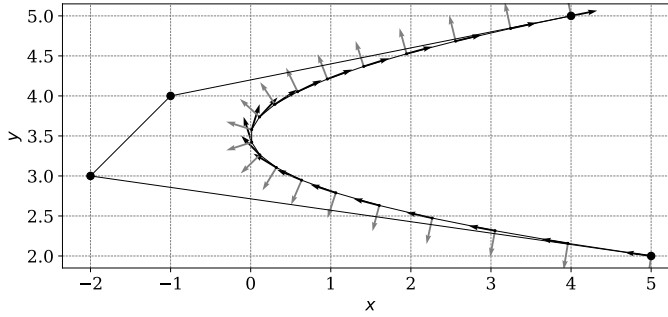

Рис. 11.6. Касательные и нормали к точкам кривой Безье третьего порядка

11.3. Задание

1. Постройте кривые Безье. Реализуйте построение как с помощью разложения по функциям Бернштейна, так и с помощью алгоритма де Кастельё. Для функций Бернштейна биномиальные коэффициенты вычислите заранее и сохраните во вложенном списке. Сравните быстродействие этих двух алгоритмов с помощью команды `%timeit`. Изобразите на графике характеристические ломаные, выделив маркерами опорные точки и саму кривую Безье.
2. Постройте все вспомогательные характеристические ломаные, которые рассчитываются алгоритмом де Кастелье.

3. Реализуйте вычисление кривой Безье второго и третьего порядков с помощью матричного алгоритма. Сравните его быстродействие с двумя уже реализованными в предыдущих заданиях алгоритмами вычисления. Быстродействие оцените с помощью команды `%timeit`.
4. Вычислите производные и нормали в каждой точке кривой Безье. Постройте касательную и нормаль в произвольной точке. Анимируйте перемещение касательной и нормали вдоль кривой.

В качестве справочной литературы рекомендуется использовать следующие источники [1—6].

11.4. Содержание отчета

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:
- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку задания;
- описание результатов выполнения задания:
  - полученные вами графические изображения;
  - листинги (исходный код) программ (если они есть);
- выводы, согласованные с целью работы.

11.5. Контрольные вопросы

1. Дайте определение кривым Безье. В каких областях они применяются?
2. Что такое функции Бернштейна и как они связаны с кривыми Безье?
3. Что такое опорные точки кривых Безье?
4. Запишите матричные формулы для вычисления кривых Безье второго и третьего порядков. Какое преимущество даёт такая форма записи?
5. Сформулируйте алгоритм де Кастельё. Какие преимущества он даёт при построении кривой Безье?
6. Как построить опорные ломаные для кривой Безье?
7. Как наиболее просто найти производные для каждой точки кривой Безье?
8. Как сплайны Эрмита связаны со сплайнами Безье?

Список литературы

1. *Роджерс Д.*, *Адамс А.* Математические основы машинной графики. — М. : Мир, 2001. — ISBN 5030021434.
2. *Голованов Н. Н.* Геометрическое моделирование. — М. : Физматлит, 2002. — ISBN 5940520480.
3. *Дубровин Б. А.*, *Новиков С. П.*, *Фоменко А. Т.* Современная геометрия: Методы и приложения. Т. 1. — 6-е изд. — М. : УРСС: Книжный дом «ЛИБРОКОМ», 2013. — ISBN 978-5-453-00047-0.
4. *Фиников С. П.* Курс дифференциальной геометрии. — М. : URSS, 2017.
5. Python home site. — 2018. — URL: `https://www.python.org/`.
6. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Лабораторная работа № 12. Сплайн Катмулла–Рома

12.1. Цель лабораторной работы

Целью данной лабораторной работы является изучение основных свойств сплайнов Катмулла–Рома.

12.2. Предварительные сведения

Сплайны Катмулла–Рома — это семейство кубических интерполяционных сплайнов, отличающихся той особенностью, что касательная в каждой точке вычисляется с использованием предыдущей и последующей точек.

Сплайны названы в честь Эдвина Катмулла (Edwin Catmull, 1945) и Рафаэля Рома (Raphael Rom). Эдвин Катмулл, по образованию компьютерный инженер, работал в студии спецэффектов Industrial Light and Magic (основатель Джордж Лукас), в Walt Disney Animation Studios и Pixar. В настоящее время является президентом последних двух анимационных студий.

Для вычисления сплайнов следует воспользоваться следующей матричной формулой:

$$\mathbf{r}(t) = \begin{pmatrix} 1 & t & t^2 & t^3 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 & 0 \\ -\tau & 0 & \tau & 0 \\ 2\tau & \tau-3 & 3-2\tau & -\tau \\ -\tau & 2-\tau & \tau-2 & \tau \end{pmatrix} \begin{pmatrix} \mathbf{p}_{i-2} \\ \mathbf{p}_{i-1} \\ \mathbf{p}_i \\ \mathbf{p}_{i+1} \end{pmatrix}.$$

Сплайн Катмулла–Рома имеет непрерывную производную во всех точках, но лежит вовне своей характеристической ломаной. Параметр τ называется *параметром натяжения*, и от него зависит, насколько резко кривая огибает контуры опорных точек.

Важно отметить, что из четырёх опорных точек $\mathbf{p}_{i-2}$, $\mathbf{p}_{i-1}$, $\mathbf{p}_i$ и $\mathbf{p}_{i+1}$ первая и последняя игнорируются, и кривая строится только от $\mathbf{p}_{i-1}$ до $\mathbf{p}_i$.

На рис. 12.1 приведён пример сплайна Катмулла–Рома для 5 опорных точек.

На рис. 12.2 приведён пример интерполяции функции sin с помощью сплайнов Катмулла–Рома.

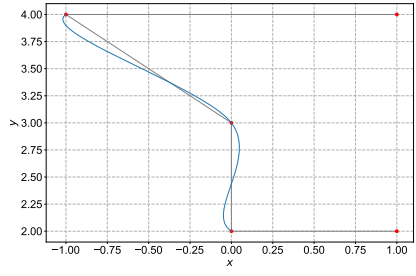

Рис. 12.1. Сплайн Катмулла–Рома для 5 опорных точек

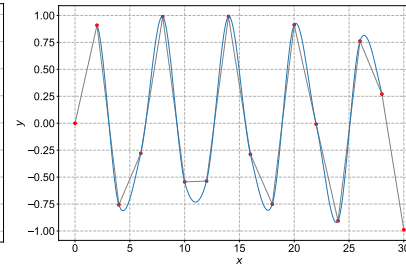

Рис. 12.2. Интерполяция функции sin **с помощью сплайнов Катмулла–Рома**

12.3. Задание

1. Постройте один сплайн Катмулла–Рома по произвольным опорным точкам. Постройте несколько изображений с разным параметром натяжения τ, как показано на рис. 12.3.

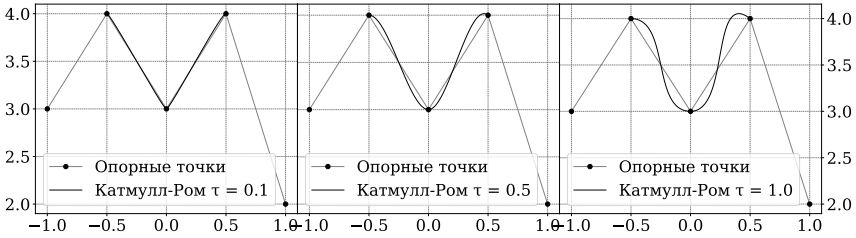

Рис. 12.3. Сплайны Катмулла–Рома с разным коэффициентом натяжения τ

2. Создайте анимацию, которая показывает как изменяется сплайн Катмулла–Рома при движении одной из опорных точек.
3. Интерполируйте произвольную гладкую функцию сплайнами Катмулла–Рома (см. пример на рис. 12.4).

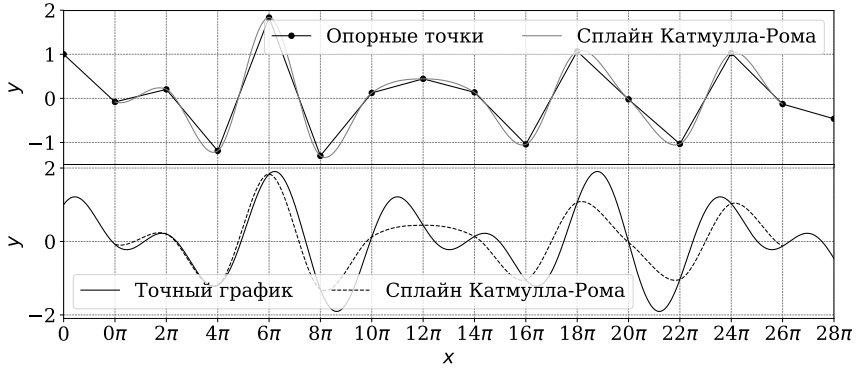

Рис. 12.4. Пример интерполяции сплайном Катмулла–Рома функции
$y(x) = \sin(x) + \cos(1.5x)$

В качестве справочной литературы рекомендуется использовать следующие источники [1—6].

12.4. Содержание отчёта

Выполнение лабораторной работы необходимо оформить в виде отчёта, который должен содержать:

- титульный лист с указанием номера лабораторной работы и ФИО студента;
- формулировку задания;
- описание результатов выполнения задания:
  - полученные вами графические изображения;
  - листинги (исходный код) программ (если они есть);
- выводы, согласованные с целью работы.

12.5. Контрольные вопросы

1. Дайте определение сплайну Катмулла–Рома. Какие особенности отличают его от ранее изученных сплайнов?
2. Как строится опорная ломаная для сплайнов Катмулла–Рома?
3. Как сплайны Катмулла–Рома связаны со сплайнами Эрмита?
4. Запишите формулы для сплайна Катмулла–Рома четвертого порядка.

Список литературы

1. *Роджерс Д.*, *Адамс А.* Математические основы машинной графики. — М. : Мир, 2001. — ISBN 5030021434.
2. *Голованов Н. Н.* Геометрическое моделирование. — М. : Физматлит, 2002. — ISBN 5940520480.
3. *Дубровин Б. А.*, *Новиков С. П.*, *Фоменко А. Т.* Современная геометрия: Методы и приложения. Т. 1. — 6-е изд. — М. : УРСС: Книжный дом «ЛИБРОКОМ», 2013. — ISBN 978-5-453-00047-0.
4. *Фиников С. П.* Курс дифференциальной геометрии. — М. : URSS, 2017.
5. Python home site. — 2018. — URL: `https://www.python.org/`.
6. *Плас Д. В.* Python для сложных задач. Наука о данных и машинное обучение. — М. : Питер, 2018. — ISBN 978-5-496-03068-7.

Учебно-методический комплекс

Рекомендуется для направления подготовки

02.03.01 — «Математика и компьютерные науки»

Квалификация (степень) выпускника: бакалавр

Программа дисциплины

1. Цели и задачи дисциплины

Целью дисциплины является овладение основным аппаратом компьютерной геометрии и элементами геометрических численных методов (численное моделирование динамических систем).

2. Место дисциплины в структуре ОП ВО

Дисциплина «Компьютерная геометрия и геометрическое моделирование» относится к вариативной части блока «Дисциплины (модули)» учебного плана по направлению 02.03.01 «Математика и компьютерные науки».

В табл. П.1 приведены предшествующие и последующие дисциплины, направленные на формирование компетенций обучающегося.

Таблица П.1

Предшествующие и последующие дисциплины, направленные на формирование компетенций по направлению 02.03.01

№ п/п	Шифр компетенции	Предшествующие дисциплины	Последующие дисциплины (группы дисциплин)
Общекультурные компетенции			
1.	—	—	—
Общепрофессиональные компетенции			
1.	ОПК-1, ОПК-4	Математический анализ, Аналитическая геометрия, Фундаментальная и компьютерная алгебра, Дифференциальные уравнения, Дифференциальная геометрия и топология, Вычислительный эксперимент и методы вычислений, Математическое моделирование	ВКР, ГОС
Профессиональные компетенции — производственно-технологическая деятельность			
1.	—	—	—
Профессионально-специализированные компетенции специализации			
1.	—	—	—

Описание компетенций для направления 02.03.01:

ОПК-1 — готовность использовать фундаментальные знания в области математического анализа, комплексного и функционального анализа, алгебры, аналитической геометрии, дифференциальной геометрии и топологии, дифференциальных уравнений, дискретной математики и математической логики, теории вероятностей, математической статистики и случайных процессов, численных методов, теоретической механики в будущей профессиональной деятельности;

ОПК-4 — способность находить, анализировать, реализовывать программно и использовать на практике математические алгоритмы, в том числе с применением современных вычислительных систем.

3. Требования к результатам освоения дисциплины

В результате изучения дисциплины студент должен:

Знать:

– основные понятия, обозначения и методы компьютерной геометрии и топологии;
– формулировки утверждений и методы их доказательства;
– корректные постановки классических задач компьютерной геометрии и геометрического моделирования;
– основные области приложения компьютерной геометрии и геометрического моделирования.

Уметь:

– решать задачи практического характера в компьютерной геометрии и геометрическом моделировании;
– применять программные средства и библиотеки для решения задач из данной предметной области;
– уметь грамотно пользоваться языком предметной области;
– ориентироваться в постановках классических задач предметной области;
– применять методы компьютерной геометрии и геометрического моделирования для решения задач;
– ориентироваться в основных разделах компьютерной геометрии и геометрического моделирования и уметь анализировать и синтезировать информацию по данному предмету, полученную из разных источников.

Владеть:

– математическим аппаратом компьютерной геометрии и геометрического моделирования;
– умением видеть прикладной аспект в решении задач дифференциальной геометрии;
– методами решения задач и доказательства утверждений в этой области.

4. Объем дисциплины и виды учебной работы

Общая трудоёмкость дисциплины составляет 3 зачётные единицы.

Вид учебной работы	**Всего часов**	**Семестры**
		7
Аудиторные занятия (всего)	**51**	**51**
В том числе:		
Лекции	17	17
Практические занятия (ПЗ)	-	-
Семинары (С)	-	-
Лабораторные работы (ЛР)	34	34
Самостоятельная работа (всего)	**57**	**57**
Общая трудоёмкость: **час.** **зач. ед.**	 **108** **3**	 **108** **3**

5. Содержание дисциплины

5.1 Содержание разделов дисциплины

Раздел 1. Комплексное использование python, ipython notebook, numpy, scipy, matplotlib, sympy

Введение в python. Основные конструкции языка. iPython и iPython notebook. Дальнейшее введение в python. Списки, словари, строки, функции. Библиотека numpy и scipy. Основные функции и типы данных. Библиотека Matplotlib. Основные понятия, функции и объекты. Комплексное использование Python, iPython notebook, NumPy, SciPy, Matplotlib, SymPy. Интерактивные виджеты iPython.

Раздел 2. Преобразование координат. Кривые второго порядка

Математическая модель кривой линии в пространстве и на плоскости. Аналитические линии. Плоские кривые. Конические сечения: эллипс, гипербола, парабола. Параметрические уравнения. Уравнения радиус-векторов. Общее уравнение для кривых второго порядка. Примеры пространственных кривых. Спираль. Преобразования координат. Инварианты кривых второго порядка при преобразовании координат. Интерполяция функций и данных. Интерполяция полиномами. Феномен Рунге при интерполяции полиномами. Ломаная линия. Интерполяция ломаными. Сплайны.

Раздел 3. Сплайны

Сплайны Лагранжа, Ньютона, Эрмита. Кубические сплайны.

Раздел 4. Кривые Безье

Кривые Безье. Базис Бернштейна. Алгоритм де Кастелье. Полиномы Бернштейна. Представление кривых второго порядка кривыми Безье. Рациональные кривые Безье. B-кривые и B-сплайны.

Раздел 5. Геометрические численные методы.
Геометрическое моделирование. Учёт геометрических свойств при численном решении систем ОДУ. Геометрические численные методы. Основные подходы к построению этих методов. Геометрические методы в теории систем ОДУ. Динамические системы и их инварианты. Численные методы Рунге–Кутты. Таблица Бутчера. Порядок метода, стадийность метода и условия порядка на коэффициенты метода. Методы Рунге–Кутты–Нюстрёма. Примеры методов Рунге–Кутты и методов Рунге–Кутты–Нюстрёма. Симплектическая форма и условие симплектичности. Условие симплектичности методов типа Рунге–Кутты. Симплектические методы типа Рунге–Кутты. Примеры использования. Сохранение инвариантов симплектическими методами.

5.2 Разделы дисциплин и виды занятий

№ п/п	Наименование раздела дисциплины	Лекц.	Практ. зан.	Лаб. зан.	Семин.	СРС	Всего час.
1.	Основы Python, NumPy и Matplotlib	3		6		10	22
2.	Углубленное знакомство с Matplotlib	3		6		10	22
3.	Сплайны Ньютона, Лагранжа и кубический сплайн	3		6		10	22
4.	Сплайн Эрмита	3		6		10	22
5.	Кривые Безье	3		6		10	22
6.	Сплайн Катмулла–Рома	2		4		7	15
Итого:		**17**		**34**		**57**	**108**

6. Лабораторный практикум

К каждому разделу полагается одна лабораторная работа. Таким образом номер лабораторной работы соответствует разделу.
1. Основы Python, NumPy и Matplotlib.
2. Углубленное знакомство с Matplotlib.
3. Сплайны Ньютона, Лагранжа и кубический сплайн.
4. Сплайн Эрмита.
5. Кривые Безье.
6. Сплайн Катмулла–Рома.

7. Практические занятия (семинары)

Практические занятия (семинары) не предусмотрены.

8. Материально-техническое обеспечение дисциплины

Дисплейные классы (ДК3, ДК4, ДК6, расположенные по адресу: г. Москва, ул. Орджоникидзе, д. 3, корп. 5) с компьютерными рабочими местами пользователей с процессором не ниже Intel Core i3-550 3.2 GHz.

9. Информационное обеспечение дисциплины

а) Программное обеспечение: GNU/Linux, Jupyter, Python 3, библиотеки numpy и matplotlib

б) Базы данных, информационно-справочные и поисковые системы: доступ к официальным сайтам и документации используемого программного обеспечения

10. Учебно-методическое обеспечение дисциплины

а) Основная литература:

- Плас Дж. Вандер. Python для сложных задач. Наука о данных и машинное обучение. — Москва : Питер, 2018.—ISBN: 978-5-496-03068-7.
- Любанович Билл. Простой Python. Современный стиль программирования. —Москва : Питер, 2017.—ISBN: 978-5-496-02088-6
- Голованов Н.Н. Геометрическое моделирование. — Москва: издательский центр «Академия», 2011. — 272 с.— ISBN 978-5-7695-7168-8.

б) Дополнительная литература:

- Мищенко А. С., Фоменко А. Т. Краткий курс дифференциальной геометрии и топологии. — Москва: ФИЗМАТЛИТ, 2004. — 304 с. — ISBN: 5-9221-0442-X.
- Дубровин Б. А., Новиков С. П., Фоменко А. Т. Современная геометрия: Методы и приложения. — 6-е изд.— М.: УРСС: Книжный дом «ЛИБРОКОМ», 2013. — Т. 1: Геометрия поверхностей, групп преобразований и полей. — 336 с.— ISBN: 978-5-453-00047-0.
- Дубровин Б. А., Новиков С. П., Фоменко А. Т. Современная геометрия: Методы и приложения. — 6-е изд.— М.: УРСС: Книжный дом «ЛИБРОКОМ», 2013. — Т. 2: Геометрия и топология многообразий. — 304 с.— ISBN: 978-5-453-00048-7.
- Рашевский П. К. Риманова геометрия и тензорный анализ. — М.: УРСС, 2014. — Т. 1: Евклидовы пространства и аффинные пространства. Тензорный анализ. Математические основы специальной теории относительности. — 352 с.— ISBN: 978-5-396-00577-8.
- Рашевский П. К. Риманова геометрия и тензорный анализ. — М.: УРСС, 2014. — Т. 2: Римановы пространства и пространства аффинной связности. Тензорный анализ. Математические основы общей теории относительности. — 336 с.— ISBN: 978-5-396-00578-5.

11. Методические указания для обучающихся по освоению дисциплины

Учебным планом на изучение дисциплины отводится один семестр. В дисциплине предусмотрены лекции, лабораторный практикум, контрольные мероприятия. В конце семестра проводится итоговый контроль знаний.

11.1. Методические указания по самостоятельному освоению теоретического материала по дисциплине

Лекционный материал дисциплины охватывает темы, указанные в разделе 5.1 программы дисциплины. В ТУИС (`http://esystem.pfur.ru`) по темам лекций размещены презентации. Рекомендуется по указанным темам в дополнение к презентациям изучить литературу, указанную в п. 10 программы дисциплины.

11.2. Методические указания по освоению дисциплины

- Продолжительность обучения: 01 сентября – 30 декабря.
- Курс предполагает 2 часа практических занятий в неделю.
- Курс предполагает большое количество самостоятельной работы, методические рекомендации по работе и необходимые материалы с содержанием курса находятся внутри модуля.

Условия прохождения курса.

Студенты набирают баллы, выполняя 6 лабораторных работ (16, 18, 16, 16, 18, 16 баллов соответственно). Итоговая оценка ставится по набранным за семестр баллам.

Порядок сдачи лабораторных работ

- Для каждой лабораторной необходимо написать все требуемые программы, а также провести требуемые тесты производительности и оформить результаты в виде таблиц и графиков.
- Для получения полных баллов необходимо защитить лабораторные, ответив на вопросы преподавателя и показав графики/таблицы с результатами работы программ.
- В случае сдачи без защиты необходимо оформить отчет и загрузить его через портал. В отчёте должны быть представлены скриншоты запуска программы, результаты их работы, а также таблицы и графики, если они требуются. В случае сдачи отчёта без дальнейшей защиты ставиться лишь 70% от полного балла за лабораторную.
- Сроки сдачи указаны для каждой лабораторной работы.
- В случае сдачи лабораторной не в срок, но с защитой ставится не более 70% от максимального балла. Если сдача не в срок и без защиты, то ставится не более 50% от максимального балла.

Паспорт фонда оценочных средств

Код компетенции	Контролируемый раздел	Контролируемая тема	ФОСы			Баллы темы	Баллы раздела
			Тек. контр.		Пром. атт.		
			ЛР	ДЗ	Экзамен/ зачет		
ОПК-1, ОПК-4	Вводные сведения и освоение инструментов	Совместное использование Python, NumPy, Matplotlib	11	5	По сумме набранных баллов	16	34
ОПК-1, ОПК-4		Преобразование координат. Кривые второго порядка	13	5		18	
ОПК-1, ОПК-4	Интерполяция сплайнами	Сплайны Лагранжа, Ньютона и кубические сплайны	11	5		16	66
		Сплайны Эрмита	11	5		16	
		Кривые Безье	13	5		18	
		Сплайны Катмулла–Рома	11	5		16	
Итого:			70	30		100	100

Описание компетенций для направления 02.03.01:

ОПК-1 — готовность использовать фундаментальные знания в области математического анализа, комплексного и функционального анализа, алгебры, аналитической геометрии, дифференциальной геометрии и топологии, дифференциальных уравнений, дискретной математики и математической логики, теории вероятностей, математической статистики и случайных процессов, численных методов, теоретической механики в будущей профессиональной деятельности;

ОПК-4 — способность находить, анализировать, реализовывать программно и использовать на практике математические алгоритмы, в том числе с применением современных вычислительных систем.

Фонд оценочных средств

Балльно-рейтинговая система оценки уровня знаний

Сводная оценочная таблица дисциплины

Код компетенции	Контролируемый раздел	Контролируемая тема	ФОСы			Баллы темы	Баллы раздела
			Тек. контр.		Пром. атт.		
			ЛР	ДЗ	Экзамен/ зачет		
ОПК-1, ОПК-4	Вводные сведения и освоение инструментов	Совместное использование Python, NumPy, Matplotlib	11	5	По сумме набранных баллов	16	34
ОПК-1, ОПК-4		Преобразование координат. Кривые второго порядка	13	5		18	
ОПК-1, ОПК-4	Интерполяция сплайнами	Сплайны Лагранжа, Ньютона и кубические сплайны	11	5		16	66
		Сплайны Эрмита	11	5		16	
		Кривые Безье	13	5		18	
		Сплайны Катмулла–Рома	11	5		16	
Итого:			70	30		100	100

Таблица соответствия баллов и оценок

Баллы БРС	Традиционные оценки РФ	Оценки ECTS
95–100	5	A
86–94		B
69–85	4	C
61–68	3	D
51–60		E
31–50	2	FX
0–30		F
51–100	Зачёт	Passed

Правила применения БРС

1. Раздел (тема) учебной дисциплины считаются освоенными, если студент набрал более 50 % от возможного числа баллов по этому разделу (теме).
2. Студент не может быть аттестован по дисциплине, если он не освоил все темы и разделы дисциплины, указанные в сводной оценочной таблице дисциплины.
3. По решению преподавателя и с согласия студентов, не освоивших отдельные разделы (темы) изучаемой дисциплины, в течение учебного семестра могут быть повторно проведены мероприятия текущего контроля успеваемости или выданы дополнительные учебные задания по этим темам или разделам. При этом студентам за данную работу засчитывается минимально возможный положительный балл (51 % от максимального балла).
4. При выполнении студентом дополнительных учебных заданий или повторного прохождения мероприятий текущего контроля полученные им баллы засчитываются за конкретные темы. Итоговая сумма баллов не может превышать максимального количества баллов, установленного по данным темам (в соответствии с приказом Ректора № 564 от 20.06.2013). По решению преподавателя предыдущие баллы, полученные студентом по учебным заданиям, могут быть аннулированы.
5. График проведения мероприятий текущего контроля успеваемости формируется в соответствии с календарным планом курса. Студенты обязаны сдавать все задания в сроки, установленные преподавателем.
6. Время, которое отводится студенту на выполнение мероприятий текущего контроля успеваемости, устанавливается преподавателем. По завершении отведённого времени студент должен сдать работу преподавателю, вне зависимости от того, завершена она или нет.
7. Использование источников (в том числе конспектов лекций и материалов семинаров) во время выполнения контрольных мероприятий возможно только с разрешения преподавателя.
8. Отсрочка в прохождении мероприятий текущего контроля успеваемости считается уважительной только в случае болезни студента, что подтверждается наличием у него медицинской справки, заверенной круглой печатью в поликлинике № 25, предоставляемой преподавателю не позднее двух недель после выздоровления. В этом случае выполнение контрольных мероприятий осуществляется после выздоровления студента в срок, назначенный преподавателем. В противном случае, отсутствие студента на контрольном мероприятии признается неуважительным.
9. Если в итоге за семестр студент получил менее 31 балла, то ему выставляется оценка F и студент должен повторить эту дисциплину в установленном порядке. Если же в итоге студент получил 31–50 баллов, т. е. FX, то студенту разрешается добор необходимого (до 51) количества баллов путём повторного одноразового выполнения предусмотренных контрольных мероприятий, при этом по усмотрению преподавателя аннулируются соответствующие предыдущие результаты. Ликвидация задолженностей проводится в период с 07.02 по 28.02 (с 07.09 по 28.09) по согласованию с деканатом.

Критерии оценки по дисциплине

95-100 баллов:

– полное и своевременное выполнение на высоком уровне лабораторных работ с оформлением отчётов, успешное прохождение контрольных мероприятий, предусмотренных программой курса;

– систематизированное, глубокое и полное освоение навыков и компетенций по всем разделам программы дисциплины;
– использование научной терминологии, стилистически грамотное, логически правильное изложение ответов на вопросы, умение делать обоснованные выводы;
– безупречное владение программным обеспечением, умение эффективно использовать его в постановке и решении научных и профессиональных задач;
– выраженная способность самостоятельно и творчески решать поставленные задачи;
– полная самостоятельность и творческий подход при изложении материала по программе дисциплины;
– полное и глубокое усвоение основной и дополнительной литературы, рекомендованной программой дисциплины и преподавателем.

86–94 балла:

– полное и своевременное выполнение на хорошем уровне лабораторных работ с оформлением отчётов, успешное прохождение контрольных мероприятий, предусмотренных программой курса;
– систематизированное, глубокое и полное освоение навыков и компетенций по всем разделам программы дисциплины;
– использование научной терминологии, стилистически грамотное, логически правильное изложение ответа на вопросы, умение делать обоснованные выводы;
– хорошее владение программным обеспечением, умение эффективно использовать его в постановке и решении научных и профессиональных задач;
– способность самостоятельно решать поставленные задачи в нестандартных производственных ситуациях;
– усвоение основной и дополнительной литературы, нормативных и законодательных актов, рекомендованных программой дисциплины и преподавателем.

69–85 баллов:

– своевременное выполнение на хорошем уровне лабораторных работ с оформлением отчётов, прохождение контрольных мероприятий, предусмотренных программой курса;
– хороший уровень культуры исполнения лабораторных работ;
– систематизированное и полное освоение навыков и компетенций по всем разделам программы дисциплины;
– владение программным обеспечением, умение использовать его в постановке и решении научных и профессиональных задач;
– способность самостоятельно решать проблемы в рамках программы дисциплины;
– усвоение основной литературы.

51-68 баллов:

– выполнение на удовлетворительном уровне лабораторных работ с оформлением отчётов, прохождение контрольных мероприятий, предусмотренных программой курса;
– систематизированное и полное освоение навыков и компетенций по всем разделам программы дисциплины;
– удовлетворительное владение программным обеспечением, умение использовать его в постановке и решении научных и профессиональных задач;
– способность решать проблемы в рамках программы дисциплины;
– удовлетворительное усвоение основной литературы;

31–50 баллов, НЕ ЗАЧТЕНО:

– невыполнение, несвоевременное выполнение или выполнение на неудовлетворительном уровне лабораторных работ, непрохождение контрольных мероприятий, предусмотренных программой курса;
– недостаточно полный объем навыков и компетенции в рамках программы дисциплины;

– неумение использовать в практической деятельности научной терминологии, изложение ответа на вопросы с существенными стилистическими и логическими ошибками;
– слабое владение программным обеспечением по разделам программы дисциплины, некомпетентность в решении стандартных (типовых) производственных задач;
– способность решать проблемы в рамках программы дисциплины;
– удовлетворительное усвоение основной литературы.

0-30 баллов, НЕ ЗАЧТЕНО:

– отсутствие умений, навыков, знаний и компетенций в рамках программы дисциплины;
– невыполнение лабораторных заданий, непрохождение контрольных мероприятий, предусмотренных программой курса; отказ от ответов по программе дисциплины;
– игнорирование занятий по дисциплине по неуважительной причине.

Примерный перечень оценочных средств

п/п	Наименование оценочного средства	Краткая характеристика оценочного средства	Представление оценочного средства в фонде
Аудиторная работа			
1.	Опрос	Средство контроля, организованное как специальная беседа преподавателя с обучающимся на темы, связанные с изучаемой дисциплиной, и рассчитанное на выяснение объёма знаний обучающегося по определённому разделу, теме, проблеме и т.п.	Вопросы по темам/разделам дисциплины
2.	Лабораторная работа	Система практических заданий, направленных на формирование практических навыков у обучающихся	Фонд практических заданий
Самостоятельная работа			
3.	Выполнение домашних заданий	В качестве домашних заданий предлагаются лабораторные работы. Объем лабораторных работ подразумевает самостоятельную работу студента в отведённые для этого учебным планом часы	Комплект разноуровневых задач и заданий

Учебным планом на изучение дисциплины отводится один семестр. В дисциплине предусмотрены лекции, лабораторный практикум, контрольные мероприятия по проверке отчётов по лабораторным работам. В конце выставляется итоговая оценка по сумме набранных баллов.

Комплект заданий для итогового контроля знаний

Итоговая оценка за курс выставляется по сумме набранных баллов за лабораторные работы и доклад. Для учащихся, которые набрали от 31 до 50 баллов, проводится контроль знаний в форме устного или письменного опроса, в рамках которого за ответы на вопросы обучающиеся могут набрать недостающее до проходного значения число баллов (51 балл).

Примерный перечень вопросов итогового контроля знаний:

1. Что изучает компьютерная геометрия? Чем она отличается от компьютерной (машинной) графики?
2. Математический аппарат каких теоретических разделов математики используется в компьютерной геометрии?
3. Дайте определение плоской и трёхмерной кривой, которое используется в компьютерной геометрии.
4. Какие три основных способа записи уравнений, задающих кривую на плоскости и в пространстве? Какой из них наиболее часто используется при построении кривых с помощью компьютера?
5. Что такое натуральный параметр кривой? Как его найти? Каков его геометрический смысл?
6. Какие векторы составляют репер Френе в двумерном и трёхмерном случае?
7. Что такое кривизна кривой? Что такое центр кривизны? Радиус кривизны?
8. Что такое резольвента и эволюта? Как они связаны?
9. Что такое сплайн и чем он отличается от полинома?
10. Какие сложности могут проявиться при интерполяции полиномами высокого порядка? Как этих сложностей избежать?
11. Запишите параметрическое уравнение для ломаной линии.
12. Запишите формулы для кубического сплайна. Чем кубический сплайн отличается от кубического полинома?
13. Запишите формулы для сплайнов Ньютона и Лагранжа. Чем данные сплайны отличаются от одноимённых полиномов?
14. Запишите формулы для сплайна Эрмита. Какие формы записи его коэффициентов вы знаете?
15. Каковы преимущества и недостатки сплайнов Эрмита по сравнению со сплайнами Ньютона, Лагранжа и кубическими сплайнами?
16. Чем сплайн Эрмита отличается от одноимённых полиномов?
17. Дайте определение кривым Безье. В каких областях они применяются?
18. Что такое функции Бернштейна и как они связаны с кривыми Безье?
19. Что такое опорные точки кривых Безье?
20. Запишите матричные формулы для вычисления кривых Безье второго и третьего порядков. Какое преимущество даёт такая форма записи?
21. Сформулируйте алгоритм де Кастельё. Какие преимущества он даёт при построении кривой Безье?
22. Как построить опорные ломаные для кривой Безье?
23. Как наиболее просто найти производные для каждой точки кривой Безье?
24. Как сплайны Эрмита связаны со сплайнами Безье?
25. Дайте определение сплайну Катмулла–Рома. Какие особенности отличают его от ранее изученных сплайнов?
26. Как строится опорная ломаная для сплайнов Катмулла–Рома?
27. Как сплайны Катмулла–Рома связаны со сплайнами Эрмита?
28. Запишите формулы для сплайна Катмулла–Рома четвёртого порядка.
29. Для чего предназначена библиотека `Matplotlib`?

30. Какие преимущества есть у библиотеки `Matplotlib` по сравнению с конкурентами?
31. Какая магическая команда позволяет встраивать графики, построенные с помощью `Matplotlib` прямо в интерактивный блокнот `Jupyter`?
32. Как можно настроить внешний вид графиков, используемый `Matplotlib` по умолчанию?
33. Сколько осей координат может быть на одном изображении?
34. Что такое субграфик в терминах `Matplotlib`?
35. Какая функция является основной для построении плоских кривых?
36. Какие методы позволяют создавать сетки субкоординат (субграфиков)?
37. Какие параметры можно передавать функции `plot` для настройки внешнего вида строящихся кривых? Перечислите некоторые из них.
38. Как в `Jupyter` получить доступ к строке документации той или иной функции?
39. Какой метод следует вызвать, чтобы отобразить легенду для построенных графиков?
40. Какие настройки есть у легенды? Как добавить заголовок легенды? Как изменить её расположение на рисунке?
41. Какие методы позволяют установить подписи к осям и заголовок?
42. Каким образом в текстовые элементы, располагаемые на осях, можно добавить формулы в формате LaTeX?
43. Какие настройки регулируют цвет, размер и форму маркеров? Как настроить частоту расстановки маркеров на кривой?
44. Какие методы регулируют отображение символов разметки осей координат?
45. Какие методы позволяют манипулировать осями координат и регулировать их видимость?
46. Как построить график в полярной системе координат?
47. Что такое примитивы?
48. Какие примитивы есть в `Matplotlib`?
49. Какой метод позволяет добавить созданные примитивы на график?
50. Для чего нужен метод `Line2D`?
51. Какой метод позволяет добавлять на координатную плоскость текстовые метки?
52. Какие параметры, регулирующие внешний вид текста на координатной плоскости, вы знаете?
53. Чем аннотации отличаются от обычного текста?
54. Может ли метод `annotate` заменить метод `text`?
55. Как настраиваются элементы аннотаций?
56. Какая функция позволяет визуализировать векторное поле?
57. Какая функция позволяет построить ступенчатый график?
58. Как в `Matplotlib` проще всего построить горизонтальную и вертикальную прямую линию?
59. Для чего нужна утилита `ffmpeg` и как её использовать в связке с `Matplotlib`?
60. Пользуясь дополнительными источниками и официальной документацией, ответьте на следующие вопросы:
    - Какая функция `Matplotlib` позволяет строить гистограммы?
    - Как установить логарифмический масштаб вдоль одной из осей координат?
    - Как минимизировать поля при сохранении созданного изображения в файл?
    - Какие функции `Matplotlib` позволяют строить трёхмерные графики?
    - Есть ли в `Matplotlib` встроенные средства создания анимированных графиков?
    - Какие недостатки `Matplotlib` вы можете выделить после знакомства с этой библиотекой?
61. Дайте краткую характеристику библиотеке `NumPy`.

62. Какое стандартное сокращение принято использовать при импорте библиотеки `NumPy`?
63. Какие недостатки встроенного в `Python` типа `list` призван устранить массив `NumPy`?
64. Какие типы данных поддерживает `ndarray`?
65. Какие функции для создания массивов вы знаете?
66. В каких случаях стоит предпочесть `ndarray` встроенному списку `Python`?
67. Как создать многомерный массив?
68. Массивы какого вида можно создавать с помощью специальных функций `eye`, `ones`, `zeros` и `empty`?
69. Как явно указать тип элементов массива?
70. На каком языке написаны большинство функций `NumPy`?
71. Что означает термин «векторизация» в контексте библиотеки `NumPy`?
72. Что такое универсальные функции и как их использование может ускорить программу?
73. Какие две функции позволяют создать линейную последовательность элементов?
74. Как получить доступ к элементам массива с помощью индексов? С какого индекса начинается нумерация? Как получить доступ к последнему индексу?
75. Как работает синтаксис срезов?
76. Как можно эффективно сравнить все соответствующие элементы двух массивов?
77. Как перебрать все элементы массива пользуясь циклом, но при этом не использовать индексы?
78. Как работают функции `concatenate`, `vstack`, `hstack`?
79. Как работают функции `split`, `split`, `hsplit`?
80. Какие математические функции определены в `NumPy`? В каких случаях их стоит применять?
81. При каких операциях `NumPy` может использовать несколько ядер процессора?
82. Какие функции позволяют вычислять среднее, сумму, кумулятивную сумму элементов массива? Могут ли они работать с многомерными массивами?
83. Какие статистические функции `NumPy` вы знаете?
84. Как создать логическую маску и как её использовать для доступа к элементам массива?
85. Какие логические операторы следует использовать для составления сложных выражений при сравнении элементов разных массивов сравнения?
86. Какая функция позволяет вычислить количество ненулевых элементов массива? Как с её помощью вычислить количество любых других одинаковых элементов?
87. Чем функции `any` и `all` из библиотеки `NumPy` отличаются от одноимённых встроенных функций?
88. Какие функции позволяют считать данные из файла? Какого формата данные они могут обрабатывать?
89. Охарактеризуйте формат CSV. Для чего его можно использовать?
90. Могут ли в массивах `NumPy` содержаться разнотипные данные?
91. Как сохранить существующий массив на диск? Какой формат данных при этом можно использовать?
92. Попробуйте сформулировать принципы, следуя которым можно добиться наиболее оптимального в смысле быстродействия использования библиотеки `NumPy`.
93. Какие функции библиотеки `NumPy` могут пригодиться при построении кривых и поверхностей?
94. Пользуясь дополнительной литературой и официальной документацией, ответьте на следующие вопросы:
    - Что такое трансляция (broadcasting) массивов? Какие правила автоматической трансляции использует `NumPy`?

   - Какие функции из библиотеки `NumPy` могут использоваться для сортировки массива?
   - Для чего нужна функция `vectorize`?
95. Кратко опишите основные особенности языка `Python`.
96. Что такое интерпретатор? Какие интерпретаторы языка `Python` вы знаете? Чем они отличаются друг от друга?
97. Назовите основные отличия языка с динамическим от языка со статическим типизированием. К какому из этих двух видов относится `Python`?
98. Какие основные типы данных языка `Python` вы знаете?
99. В чем разница между списком (`list`) и кортежом (`tuple`) в языке `Python`?
100. Какие инструкции управления потоком выполнения есть в языке `Python`?
101. Что такое списковые сборки и для чего они предназначены?
102. Для чего нужны генераторы? Не дублируют ли они функционал списков?
103. Можно ли обратиться к символу строки по индексу? Можно ли заменить произвольный символ у уже существующей строки?
104. Можно ли обратиться к элементу кортежа по индексу? Можно ли заменить произвольный элемент кортежа?
105. Могут ли списки `Python` хранить данные привольного типа?
106. Какими методами обладает список в языке `Python`?
107. Какую инструкцию следует использовать для сравнения двух переменных булева типа?
108. Какую инструкцию следует использовать для проверки вхождения подстроки в строку?
109. Каким образом можно последовательно перебрать все элементы списка? Можно ли похожим образом поступить с кортежем? Со строкой?
110. Какие функции создают список, кортеж, словарь?
111. Какие методы есть у строки?
112. Что такое словарь в языке `Python`? Какие у него есть аналоги в других языках программирования?
113. Как упорядочены элементы словаря?
114. Как можно последовательно перебрать все элементы словаря, не зная его ключей?
115. Как объявляется функция в языке `Python`?
116. Как задать обязательные аргументы функции? Необязательные аргументы?
117. Для чего служат синтаксические конструкции `*args` и `**kwargs`? Приведите пример.
118. Можно ли передавать в функцию другую функцию?
119. Можно ли возвращать из функции другую функцию? Проверьте на примере.
120. Какая специфика есть у списков при передаче их внутрь функций?
121. Используя документацию языка `Python`, ответьте на следующие вопросы:
   - Для чего нужны функции `map`, `filter` и `reduce`?
   - Для чего можно использовать встроенные функции `any` и `all`?
   - Как осуществить вывод в консоль без перевода на новую строку в конце вывода?

Комплект разноуровневых задач (заданий)

1. Задания репродуктивного уровня

В качестве заданий репродуктивного уровня предлагаются вопросы для самопроверки и обсуждения по темам курса (см. лабораторный практикум).

2. Задания реконструктивного уровня

В качестве заданий реконструктивного уровня предполагаются задания лабораторного практикума.

Критерии оценки выполнения заданий по лабораторным работам

Оценивается полнота ответа на вопросы, полнота освоения пройденного материала, умение пользоваться дополнительными источниками.

Сведения об авторах

Геворкян Мигран Нельсонович — кандидат физико-математических наук, доцент кафедры прикладной информатики и теории вероятностей РУДН.

Кулябов Дмитрий Сергеевич — доцент, доктор физико-математических наук, доцент кафедры прикладной информатики и теории вероятностей РУДН.

Демидова Анастасия Вячеславовна — кандидат физико-математических наук, доцент кафедры прикладной информатики и теории вероятностей РУДН.

Королькова Анна Владиславовна — доцент, кандидат физико-математических наук, доцент кафедры прикладной информатики и теории вероятностей РУДН.

Учебное издание

Мигран Нельсонович Геворкян, Дмитрий Сергеевич Кулябов, Анастасия Вячеславовна Демидова, Анна Владиславовна Королькова

Компьютерная геометрия и геометрическое моделирование

Редактор *И. Л. Панкратова*
Технический редактор *Н. А. Ясько*
Компьютерная вёрстка *А. В. Королькова*, *Д. С. Кулябов*

Подписано в печать . .2018 г. Формат 60×84/16. Печать офсетная.
Усл. печ. л. ___. Тираж 500 экз. Заказ № 1519.

Российский университет дружбы народов
115419, ГСП-1, г. Москва, ул. Орджоникидзе, д. 3

Типография РУДН
115419, ГСП-1, г. Москва, ул. Орджоникидзе, д. 3, тел. 952-04-41