Введение

1. Вступительные слова

В начале подготовки специализации перед командой разработчиков курса стоял выбор: либо рассказать необходимую математику по ходу изложения курса или же сделать отдельный подготовительный курс. Был принят второй вариант, а именно сперва рассказать основы линейной алгебры, математического анализа и программирования на языке Python. Причем математические концепции в данном курсе приводятся сразу на наглядных практических примерах.

2. История Python

История языка Python начинается в 1980-х годах, когда Гвидо ван Россум, сотрудник центра математики и информатики в Нидерландах, приступил к созданию его первой версии. И до сих пор «Гвидо ван Россум» остается «великодушным пожизненным диктатором»[1].

К данному моменту были выпущены уже три версии языка — Python 1, Python 2, Python 3, причем первые две из них обратно совместимы, а Python 3 и Python 2 — уже нет. Это связано с тем, что те улучшения, которые были сделаны в Python 3, невозможно внести без нарушения обратной совместимости. Официально поддерживаются обе версии: Python 3 — это будущее языка, более современная версия, а Python 2 поддерживается до сих пор из-за огромного количества наработок, которые еще не были перенесены на более свежую версию.

Следует особо отметить, что Python выпускается под свободной лицензией «Python Software Foundation License», разрешающей использовать исходный код проекта не только в открытом, но и в коммерческом программном обеспечении. Python работает почти на всех известных платформах. Существуют версии для MS Windows, Linux, Mac OS, FreeBSD и так далее.

Также привлекательной стороной Python является богатая стандартная библиотека. В нее входят, например, модуль для работы с текстом в различных кодировках и модуль для работы с регулярными выражениями. Кроме стандартной библиотеки, доступны: библиотека `matplotlib` для построения графиков функций, `pandas` — для работы с электронными таблицами и `scipy` — для научных и инженерных расчетов.

Язык Python отличается лаконичным синтаксисом, поэтому на нем и читать чужой код, и писать свой очень просто. Разработчики языка придерживаются философии, называемой «The Zen of Python», текст которой на английском языке выводится интерпретатором по команде `import this`. Ниже представлена часть текста, переведенная на русский язык:

- Красивое лучше, чем уродливое.
- Простое лучше, чем сложное.
- Читаемость имеет значение.
- Ошибки никогда не должны замалчиваться.
- Сейчас лучше, чем никогда.
- Хотя никогда зачастую лучше, чем прямо сейчас.
- Если реализацию сложно объяснить — идея плоха.
- Если реализацию легко объяснить — идея, возможно, хороша.

Python был выбран в качестве основного языка программирования для нашей специализации по многим причинам. Вот некоторые из них:

- Python — свободное ПО.
- Python доступен практически на всех платформах.
- Python прост в изучении.
- Python можно использовать в интерактивном режиме.
- Python имеет большое сообщество пользователей и разработчиков.
- Для Python доступно огромное число библиотек. При этом богатый набор функций имеется в стандартной библиотеке.

[1] Этот шуточный термин используется, чтобы обозначать главу или основателя проекта, который сохраняет за собой право окончательного решения в жизни проекта.

3. Установка Python

Установить Python не сложно, и существует огромное количество способов, как это сделать. Для целей демонстрации, установка Python в составе дистрибутива Anaconda будет произведена на операционной системе Linux.

Сначала с сайта `https://www.continuum.io/downloads` нужно скачать установочный файл для используемой Вами операционной системы и следовать указаниям. Согласно инструкции для Linux, необходимо открыть терминал и выполнить команду `bash Anaconda2-2.4.1-Linux-x86_64.sh` (точное название файла может отличаться). Вам будет предложено ознакомиться с лицензией и указать директорию для установки. Также в процессе установки будет предложено внести изменения в файл `bash.rc`, чтобы выбрать только что установленный Python в качестве используемого по умолчанию. Для того, чтобы эти изменения вступили в силу, необходимо закрыть текущую сессию (окно терминала) и запустить новую.

Запустить интерактивную оболочку IPython можно с помощью команды `ipython notebook`. В результате исполнения этой команды IPython должен открыться в новой вкладке браузера.

4. Знакомство с IPython

После запуска IPython в браузере откроется страница со списком файлов рабочего каталога. С помощью кнопки «new» можно создать новый файл типа «IPython notebook». При создании файл откроется в новой вкладке, где сразу же можно будет указать для него название, а в рабочем каталоге появляется файл с расширением `.ipynb`.

Главное меню программы интуитивно понятно. Особо стоит отметить, что существует возможность экспортировать текущую тетрадь в виде файла .py, результаты расчетов — в виде HTML или PDF. Соответствующие команды расположены в подменю «Download As» меню «File».

Интерфейс тетради IPython основан на работе с ячейками. Их можно создавать, перемещать, редактировать и исполнять находящийся в них код с помощью соответствующих команд на панели инструментов или в главном меню тетради.

Традиционно изучение языков программирования начинается с написания программы «Hello, world!». Функцию `print` можно использовать для вывода на экран переменных любого типа:

```
t = 'Hello, world!'
print t % OUT: Hello, world!
```

В IPython значение переменной или выражения также выводится на экран, даже если опустить `print`. Это, а также богатые математические возможности IPython, позволяет использовать IPython в качестве удобного калькулятора. При этом, если используется ядро Python 2, деление двух целых чисел является целочисленным делением, то есть результатом всегда является целое число. Если же или делимое, или делитель являются числом с плавающей точкой, результат также будет числом с плавающей точкой, а деление будет пониматься в обычном смысле. Например:

```
100 /12           % OUT:  8
100./12           % OUT:  8.333333333333334
round(100./12,3)  % OUT:  8.333
```

Здесь была использована функция `round`, которая используется для округления выражения в её первом аргументе с точностью до числа знаков после запятой, задаваемой её вторым аргументов.

Если требуемая математическая функция не доступна по умолчанию, существует два пути: либо реализовать нужную функцию самостоятельно, либо подключить библиотеку, в которой содержится необходимая функция, с помощью команды `from <library> import <function>`. Функцию `factorial` из библиотеки math можно добавить с помощью следующего кода:

```
from math import factorial
factorial(3) % OUT: 6
```

Кроме ячеек с кодом на Python, в тетрадь можно добавлять ячейки других типов. Для этого необходимо поменять тип ячейки. Это можно сделать в выпадающем списке в панели инструментов. Тип «Markdown» используется для набора текста на одноименном языке разметки:

- Создание заголовков производится путём помещения знака решетки перед текстом заголовка.

- Количество знаков «#» соответствует уровню заголовка.
- Внутри $$ ... $$ можно набирать математические формулы используя LaTeX.
- Более подробное описание доступно на странице википедии https://ru.wikipedia.org/wiki/Markdown.

Если отправить ячейку типа «Markdown» на исполнение, вместо кода разметки появится отформатированный текст.

На Unix-подобных операционных системах код ячейки можно выполнять в интерпретаторе командной строки `bash`. В случае однострочных команд достаточно набрать перед этой командой символ восклицательного знака, а имена переменных IPython начинать с символа доллара:

```
!echo $t
```

Многострочные инструкции командной строки набираются с помощью «bash magic»: на первой строке набирают `%%bash`, а на следующих — желаемый скрипт командной строки. На самом деле IPython предоставляет и другие «магические команды», полный список которых можно получить набрав в ячейке `%lsmagic`.

Среди команд, представленных в `%lsmagic`, есть команда, позволяющая рисовать графики прямо внутри тетради IPython:

```
%pylab inline
% OUT: Populationg the interactive namespace from numpy and matplotlib
```

Простейший пример построения графика:

```
y = range(11) % массив чисел от 0 до 10
y             % OUT: [0,1,2,3,4,5,6,7,8,9,10]
plot(y)       % ГРАФИК
```

На этом урок заканчивается. Подробнее про IPython будет рассказано на следующих уроках.

Программирование на Python

1. Типы данных в Python

Настало время познакомиться с типами данных в Python. Числовые типы `int` и `float` уже были упомянуты ранее:

```
x = 3.5
print type(x) % OUT: <type 'float'>

y = 4
print type(y) % OUT: <type 'int'>
```

С помощью функции `type` можно узнать код произвольного объекта. Строки, например, могут быть двух типов:

```
print type("abs")  % OUT: <type 'str'>
print type(u"abs") % OUT: <type 'unicode'>
```

Отличия между ними будут подробнее обсуждаться позднее.

Упорядоченный набор значений в Python можно представить с помощью списка:

```
w = [1, 2, 3]
print type(w) % OUT: <type 'list'>
```

Списки в Python очень похожи на знакомые по другим языкам программирования массивы, но, в отличие от массивов, могут содержать элементы разных типов.

Python является языком с утиной типизацией. В языках с динамической типизацией, частным случаем которой является утиная типизация, тип переменной определяется не в момент её объявления, а в момент присваивания значения. Следовательно, в разных участках кода переменная может принимать значения разных типов. В случае утиной типизации границы применимости объекта определяются не столько конкретным типом и иерархией наследования, сколько наличием и отсутствия у него определенных методов и свойств. Термин происходит от шуточного «утиного теста»: «Если нечто выглядит, плавает и крякает как утка, то это, вероятно, и есть утка».

Другой способ представить упорядоченный набор значений состоит в определении кортежа (`tuple`). В Python кортежи записываются в круглых скобках:

```
print type((1, 2, 3)) % OUT: <type 'tuple'>
```

Кортеж относится к так называемым неизменяемым типам данных, чем существенно отличается от списка. Попытка изменить кортеж или добавить в него еще один элемент приведет к возникновению ошибки. Эта особенность позволяет использовать кортежи в качестве ключей словаря.

Неупорядоченный набор значений представим в виде множеств, которые записываются в фигурных скобках:

```
print type({1, 2, 3}) % OUT: <type 'set'>
```

Еще один тип данных, словарь, позволяет хранить в себе пары вида ключ-значение. Создать словарь можно с помощью конструктора `dict()`:

```
e = dict()      % Создан пустой словарь
e['abc'] = 3.5  % Строке-ключу 'abc' поставлено в
                %    соответствие значение 3.5 типа float
```

Ключами в словаре могут быть не только строки. В качестве ключа допустимо использовать, например, числовые значения и кортежи:

```
e[3.5]       = 'abc'
e[(1, 2, 3)] = 4.5
```

Но использование списка в качестве ключа недопустимо, так как список является изменяемым.

Синтаксис для чтения данных из словаря по ключу аналогичен синтаксису для получения данных по индексу списка. Грубо говоря, ключ есть обобщение понятия индекса списка: ключ может принимать значение произвольного хэшируемого типа. К слову, список не является хэшируемым, и именно поэтому не может быть выбран в качестве ключа.

В Python словарь использует хэш-таблицы. Идея заключается в том, чтобы заменить сравнение ключей более быстрым сравнением их хэш-значений. В таком случае важно, чтобы все ключи были хэшируемы, то есть хэш-значение не менялось во время исполнения программы, а равные ключи-объекты имели одинаковое хэш-значение.

Ранее было сказано, что множества являются изменяемыми объектами, а значит не могут выступать в роли ключей. Но в качестве ключей можно использовать неизменяемые множества, которые можно получить используя функцию `frozenset()`:

```
s = frozenset(1,2,3)
e[s] = 3.76
```

Таким образом, к данному моменту были изучены основные типы данных в Python и подробно было рассказано о существенных отличиях изменяемых и неизменяемых типов. На следующем занятии будут рассмотрены основные управляющие конструкции языка Python.

2. Синтаксис языка Python

В Python оператор условного перехода выглядит привычным образом. Например, пусть дан следующий код:

```
if x:
   print "OK"
else:
   print "NOT OK"
```

Если переменная `x` была равна `True`, то исполнится код в теле инструкции `if` и на экран будет выведено ÖK̈. В противном случае, например если x определена как `False`, исполнится код в теле инструкции `else`. В Python для выделения блоков кода используются отступы: отступы инструкций в пределах одного блока должны совпадать по величине.

Другой пример. Следующий код выводит числа от 0 до 9:

```
for i in range(10):
    print i
```

При этом в `range()` можно указать начальное значение i. Например, вот этот код выводит числа от 1 до 9.

```
for i in range(1,10):
    print i
```

В данных примерах каждое число выводится на отдельной строке, поскольку `print` по умолчанию начинает новую строчку после вывода требуемого значения. Чтобы все числа выводились в одной строке, необходимо добавить запятую в конец строчки с `print` следующим образом:

```
for i in range(1,10):
    print i,
```

Теперь числа от 1 до 9 выводятся в одной строке через пробел.

На самом деле `range()` это функция, которая возвращает список натуральных чисел в определяемом её аргументами диапазоне:

```
print range(2,5)
OUT: [2,3,4]
```

Использовать `range()` совершенно не обязательно. Циклы в Python перебирают по очереди элементы списка, стоящего после `in`. Например, можно написать так:

```
for i in [2,3,4]:
    print i,
```

В Python 2 кроме функции `range()` существует и функция `xrange()`, которую также можно использовать в цикле:

```
for i in xrange(2,5):
    print i,
OUT: 2, 3, 4
```

Результат выполнения не отличается от такового при использовании `range()`. Но можно убедиться, что `range` и `xrange` возвращают объекты разных типов:

```
print type(range(2,5))
OUT: list
print type(xrange(2,5))
OUT: iter
```

Функция `xrange` возвращает не список, а так называемый генератор. В то время, как при использовании `range()` сначала создается список, а только потом происходит итерирование по этому списку, при использовании генераторов список никогда не создается и не занимает память. Это особенно актуально, когда необходимо итерировать в очень большом диапазоне значений.

```
print xrange(2,500000000)
OUT:  xrange(2,500000000)
```

В Python существует способ удобно и компактно задавать некоторые списки с помощью так называемого конструктора списка (анг. list comprehension.):

```
w = [ x ** 2 for x in range(1,11) ]
print w
OUT: [ 1, 4, 9, 16, 25, 36, 49, 64, 81, 100 ]
print type(w)
OUT: <type 'list'>
```

В этом примере создан список квадратов чисел от 1 до 10. Если необходимо построить список, например, квадратов только четных чисел из этого диапазона, конструктор допускает использование условия:

```
w = [ x ** 2 for x in range(1,11) if x % 2 == 0]
OUT: [ 4, 16, 36, 64, 100 ]
```

Аналогично выглядит конструктор генераторов:

```
w = ( x ** 2 for x in range(1,11) if x % 2 == 0 )
OUT: ( 4, 16, 36, 64, 100 )
print type(w)
OUT: <type 'generator'>
```

В этом случае весь список не будет храниться в памяти во время работы цикла.

Также в Python доступен цикл `while`, который продолжает выполнение, пока выполнено указанное условие, а также операторы `break`, который досрочно прерывает цикл, и `continue`, который начинает следующий проход цикла, минуя оставшееся тело цикла. Например:

```
s = 0
while True: % Это условие всегда выполняется
    s += 1
    if s % 2 ==0: % Через раз
        print "Continue" % вместо s будет выведено "continue"
        continue % и код ниже в таком случае не будет исполнен в этой итерации
    print s
    if s > 10: % С помощью этой конструкции
        break  % <<бесконечный цикл досрочно прерывается>>
```

Важно уметь определять свои функции. Например, следующий код представляет собой альтернативную реализацию уже известной функции `range`:

```
def myrange(a,b):
    res = [ ]
    s = a
    while s!=b:
      res.append(s)
      s+=1
    return res
```

Этот пример демонстрирует синтаксис для создания функций. На самом деле при создании функции необходимо обработать особые случаи, но это не является текущей целью.

Функции в Python являются объектами первого класса: их можно передавать в качестве аргументов, присваивать их переменным и так далее. Например, функция `map()` позволяет обрабатывать одну или несколько последовательностей с помощью переданной в качестве аргумента функции:

```
def sq(x):
    return x ** 2

print map(sq, range(10))
OUT: [ 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Функция в Python может быть определена как с помощью оператора `def`, так и с помощью лямбда-выражения. Следующие операторы эквивалентны:

```
def sq(x):
    return x ** 2

sq = lambda x: x**2
```

Тогда предыдущий код можно записать более компактно:

```
print map(lambda x: x**2, range(10))
OUT: [ 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

3. Чтение данных из файла

Часто для анализа необходимо загрузить данные, находящиеся во внешнем файле. Этот файл может быть простым текстовым документом, XML-документом или иметь любой другой пригодный для хранения данных формат. Для достижения этой цели Python предоставляет целый ряд разнообразных инструментов. Работа будет производиться в новой тетради IPython.

Самый простой способ прочитать данные из файла — использовать функцию open из стандартной библиотеки. Эта функция позволяет считать содержимое простого текстового файла. Информацию о том, какие аргументы принимает функция и какое значение возвращает можно найти в docstring этой функции. Чтобы прочитать docstring, то есть строку документации по этой функции, необходимо набрать вопросительный знак и затем имя функции:

```
?open
```

В результате внизу появляется требуемое описание функции:

```
open(name[, mode[, buffering]]) -> file object

Open a file using the file() type, returns a file object.  This is the
preferred way to open a file.  See file.__doc__ for further information.
```

По данному тексту можно понять, что первый аргумент является обязательным и имеет смысл пути к файлу с данными (если файл лежит в рабочей директории, можно просто написать имя файла). Второй аргумент определяет режим, в котором требуется открыть файл. Существуют несколько основных режимов:

```
r   --- режим чтения   ( режим по умолчанию )
w   --- режим записи   ( данные из старого файла стираются )
a   --- режим дозаписи ( новые данные будут добавлены в конец файла )
r+  --- режим чтения и записи
```

Последний аргумент сейчас не представляет интереса. Функция возвращает объект-файл, с помощью которого будет происходить взаимодействие с файлом:

```
file_obj = open('example_utf8.txt', 'r')
type(file_obj) % OUT file
```

Таким образом файл example_utf8.txt, находящийся в рабочем каталоге, будет открыт только для чтения. С помощью метода read() можно вывести на экран все содержимое файла:

```
print file_obj.read()
%   OUTPUT:
%   Привет, мир!
%   тестовый файл
%   урок ¨чтение данных в python
¨
```

Часто бывает удобно считывать файл не целиком. С помощью метода readline можно не считывать весь файл, а считать только несколько первых строк. При первом вызове метода будет возвращена первая строка, а при повторном — вторая. Также сначала нужно открыть файл для чтения заново.

```
file_obj = open('example_utf8.txt', 'r')
print file_obj.readline()
%   OUTPUT:
%   Привет, мир!
print file_obj.readline()
%   OUTPUT:
%   тестовый файл
```

На самом деле пользоваться такими функциями совершенно не обязательно. С файлом можно работать как с обычным генератором. В следующем примере строчки файла выводятся с помощью цикла for:

```
file_obj = open('example_utf8.txt')
for line in file_obj:
    print line.strip()
```

Функция strip() удаляет все пробелы в начале и конце строки, включая символы табуляции, новой строки и так далее.

Бывает необходимым сохранить строки файла в виде списка. Это можно сделать с помощью конструктора списка list(), в качестве аргумента которого будет передан объект-файл. Поскольку объект-файл можно использовать как генератор, получившийся список будет содержать все строки файла:

```
file_obj = open('example_utf8.txt')
data_list = list(file_obj)
for line in data_list: print line.strip() % Проверка, что файл считан
```

Другой способ состоит в использовании метода readlines() файл-объекта. Этот метод возвращает список из всех содержащихся в файле строк:

```
file_obj = open('example_utf8.txt')
data_list = file_obj.readlines()
for line in data_list: print line.strip() % Проверка, что файл считан
```

Оба способа дают одинаковый результат и можно выбрать любой из них.

После того, как работа с файлом закончена, файл нужно закрыть с помощью команды close(). Считается хорошим тоном закрывать файлы, работа с которыми закончена. Во-первых, так будут освобождены системные ресурсы, которые задействованы в работе с файлом. Во-вторых, после закрытия файла он не может быть по ошибке испорчен и, если он редактировался, изменения точно будут сохранены на диске. Если метод close() был вызван для открытого файла, попытка продолжить работу с ним приведет к возникновению ошибки.

```
file_obj = open('example_utf8.txt')
file_obj.close()
file_obj.read()
% ValueError: I/O operation on closed file
```

Файл, с которым производилась работа до этого момента, был в кодировке utf8. Однако часто бывает необходимым прочитать файл в другой кодировке, например KOI8-R. Эта кодировка соответствует русскоязычной кириллице. Если непосредственно использовать функцию open(), как это делалось раньше:

```
file_obj = open('example_koi_8.txt')
print file_obj.read()
%  %D0%9F%D1%80%D0%B8%D0%B2%D0%B5%D1%82%2C%20%D0%BC%D0%B8%D1%80!
%  %%0A%D1%82%D0%B5%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9%20%D1%84%D0%B0%D0%B9%D0%BB
%  0A%D1%83%D1%80%D0%BE%D0%BA%20%5C%22%D1%87%D1%82%D0%B5%D0%BD%D0%B8%D0%B5%20%D0%B4%D0%
   B0%D0%BD%D0%BD%D1%8B%D1%85%20%D0%B2%20python%5C%22
```

Для того, чтобы корректно отобразить прочитанный файл, можно использовать библиотеку codecs. Импортирование библиотеки происходит с помощью ключевого слова import. После этого файл нужно прочитать функцией open(), которая определена в библиотеке codecs. Эта функция отличается тем, что в ней можно указать такие дополнительные параметры как кодировка файла.

```
import codecs
file_obj = codecs.open('example_koi_8.txt', 'r', encoding='koi8-r')
print file_obj.read()
%   Привет, мир!
%   тестовый файл
%   урок ¨чтение данных в python
¨
```

Теперь, поскольку файл был открыт с указанием правильной кодировки, он отображается корректно. В следующем видео будет показано, как сохранять данные в файлы.

4. Запись данных в файл

В этом видео подробно разбирается, как записать данные в файл средствами Python. Задача следующая — создать несколько текстовых строк и записать их в определенный файл.

С помощью стандартной функции `open()` необходимо открыть файл для записи:

```
file_obj = open('file_to_write_in.txt', 'w')
```

Таким образом файл `file_to_write_in.txt` в рабочем каталоге создается для записи. Если файл с таким именем существовал до этого, его содержимое теряется.

Теперь создадим переменную для строки, которую требуется записать в файл:

```
string = 'строка для записи в файл\n'
```

В конце строки набран символ перевода строки. Это делается для того, чтобы при записи нескольких строк в файл они располагались на разных строчках, а не склеивались в одну. Запись производится с помощью метода `write`, который в качестве аргумента принимает строку для записи в файл. После того, как нужные данные записаны, файл нужно обязательно закрыть с помощью метода `close`:

```
file_obj.write(string)
file_obj.close()
```

Если не закрыть файл, то очень легко его испортить. Хорошим тоном считается закрывать файлы сразу же, как только работа с ними прекращена.

Проверить, что запись файла была успешна, можно с помощью вызова утилиты `cat` средствами командной строки (для Unix-подобных операционных систем):

```
!cat file_to_write_in.txt
% OUT: строка для записи в файл
```

Пусть теперь необходимо добавить к файлу еще одну строчку. Если открыть файл и записать вторую строку так, как это делалось раньше:

```
file_obj = open('file_to_write_in.txt', 'w')
second_string = 'вторая строка для записи в файл\n'
file_obj.write(second_string)
file_obj.close()
```

то окажется, что перед записью новой строки все содержимое файла было стерто:

```
!cat file_to_write_in.txt
% OUT: вторая строка для записи в файл
```

В режиме "w", если файл уже существует, его содержимое перезаписывается новым. Чтобы добавить строчку к существующему файлу необходимо использовать режим "a":

```
# Запись 3 строки
file_obj = open('file_to_write_in.txt', 'a')
third_string = 'третья строка для записи в файл\n'
file_obj.write(third_string)
file_obj.close()
```

После выполнения этих команд в конец файла будет добавлена новая строчка:

```
!cat file_to_write_in.txt
% OUT: вторая строка для записи в файл
       третья строка для записи в файл
```

Часто необходимо записать сразу целый список строк в файл. Можно использовать цикл по списку из строк и записывать каждую с помощью функции `write`. Метод `writelines` позволяет сделать то же самое быстрее: достаточно просто передать в качестве первого аргумента список строк и он будет записан в файл.

В следующем примере файл будет открыт немного по-другому. С помощью конструкции `with ... as ... :` удается избежать проблем с необходимость помнить о своевременном закрытии файла. Файл автоматически будет закрыт, как только весь код блока `with` будет выполнен.

```
digits = range(1,11)
with open('second_file_to_write_in.txt', 'w') as file_obj:
    file_obj.writelines(digit + '\n' for digit in map(str, digits))
```

Здесь с помощью функции `map` список чисел стал списком строк, а с помощью конструктора — добавлены символы новой строки. Файл, таким образом, будет иметь следующий вид:

```
!cat second_file_to_write_in.txt
% OUT: 1
       2
       3
       4
       5
       6
       7
       8
       9
       10
```

На этом занятия, посвященные работе с файлами, завершены.

Основы математического анализа

1. Понятие функции действительной переменной

Говорят, что на некотором множестве D имеется функция f со значениями из множества E, если для каждого элемента x из D по правилу f поставлен в соответствие некоторый элемент y из множества E. Другими словами, функция f отображает множество D в множество E.

При этом множество D называется областью определения, а множество E — областью значений данной функции f. В свою очередь, множество значений Y функции — это множество таких $y \in E$, что существует $x \in D$ такой, что $f(x) = y$. Множество значений Y функции, вообще говоря, может быть подмножеством области значений E.

В рамках данного курса вопросы об определении множества вещественных (действительных) чисел R рассматриваться не будут. Основное внимание же будет уделено функциям действительного переменного и их свойствам, так как они наиболее часто встречаются в практических приложениях.

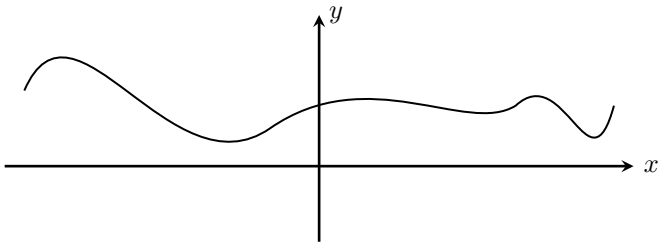

Рис. 1: График некоторой функции

Представление о функции, её свойствах и поведении можно получить, построив ее график. Графиком называется геометрическое место точек (x, y) плоскости таких, что $y = f(x)$. Функцию называют непрерывной, если, говоря нестрого, малые изменения её аргумента приводят к малым изменениям её значения. Если это условие нарушается, функция терпит разрыв.

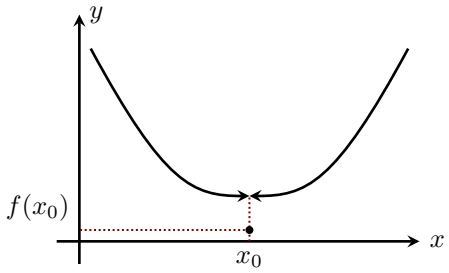

Рис. 2: Функция с устранимым разрывом

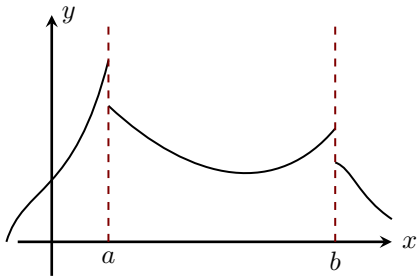

Рис. 3: Функция с разрывами в точках a и b.

Особо стоит выделить так называемые функции с устранимым разрывом в некоторой точке. В таком случае переопределение значения функции в этой точке приводит к тому, что функция становится непрерывной.

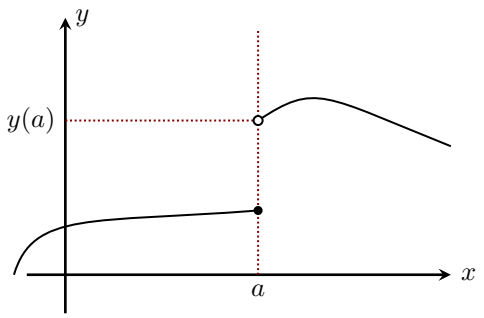

Рис. 4: Функция с разрывом типа «скачок».

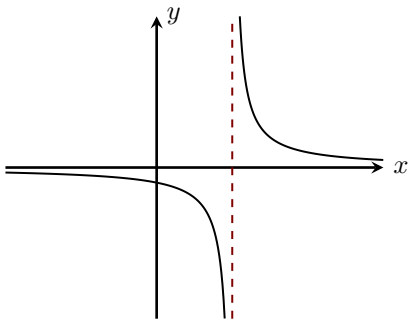

Рис. 5: Функция с бесконечным разрывом.

Другим важным случаем является разрыв типа «скачок». Также встречаются случаи, когда «функция уходит в бесконечность» в точке разрыва. На самом деле существуют еще более интересные случаи разрывов, которые выходят далеко за рамки этого курса.

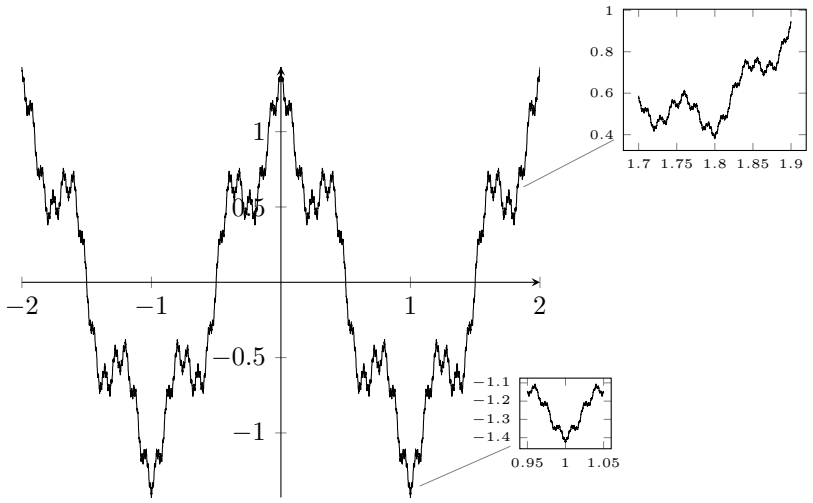

Рис. 6: Функция Вейерштрасса является непрерывной, но не имеет производной ни в одной точке.

Свойство непрерывности не является показателем того, что функция является «хорошей» в каком бы то ни было смысле. Например, функция Вейерштрасса является всюду непрерывной функцией. Но она не обладает ни в какой точке так называемым свойством гладкости, которое будет определено чуть-чуть позже.

2. Представление о пределе функции

Пусть дана функция:

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

Эта функция не определена в точке $x = 0$, но ее значение может быть вычислено в точках, сколь угодно близких к точке 0. Другими словами, будет изучаться ее поведение при $x \to 0$. Например, можно получить следующую таблицу значений для представленной функции:

x	0.1	0.01	0.001	0.0001	...
$f(x)$	2.593..	2.704..	2.716..	2.718..	...

Уже только по этим значениям можно заметить, что значение функции тем ближе к некоторой величине, чем меньше значение x. В таком случае используется обозначение:

$$\lim_{x \to 0} (1 + x)^{\frac{1}{x}} = e.$$

Однако не все функции имеют конечный предел. Например, для функции $g(x) = \frac{1}{x}$ таблица значений будет выглядеть так:

x	0.1	0.01	0.001	0.0001	...
$1/x$	10	100	1000	10000	...

Функция просто неограниченно растет при приближении x к точке 0:

$$\lim_{x \to 0} \frac{1}{x} = \infty.$$

Понятие предела оказывается неразрывно связанным с понятием непрерывности.

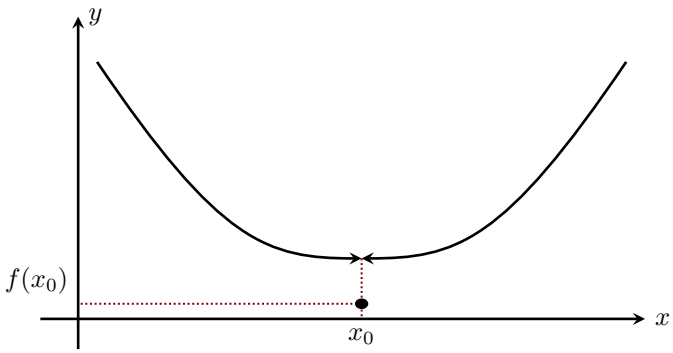

Рис. 7: Предел функции в точке устранимого разрыва не равен ее значению в этой точке.

А именно, функция $f(x)$ является непрерывной в точке $x = a$, если

$$\lim_{x \to a} f(x) = f(\lim_{x \to a} x) = f(a).$$

С помощью понятия предела определяется другое полезное понятие — понятие производной. Но прежде всего следует рассмотреть линейную функцию. Для нее легко можно получить связь коэффициента наклона и скорости роста функции:

$$\frac{f(x + \Delta x) - f(x)}{\Delta x} = k.$$

По аналогии можно ввести похожую характеристику для произвольной функции:

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x)}{\Delta x}.$$

Предельный переход позволяет обобщить понятие скорости роста на случай произвольных функций. Эта величина и называется производной функции. Теперь можно сформулировать определение гладкой функции. Гладкой называется такая функция, производная которой существует и непрерывна.

3. Геометрический смысл производной

Уравнение произвольной прямой может быть представлено в виде:

$$y = kx + b,$$

где $k = \operatorname{tg} \alpha$ — коэффициент наклона прямой, а α — угол наклона прямой (определяется как угол между прямой и осью Ox).

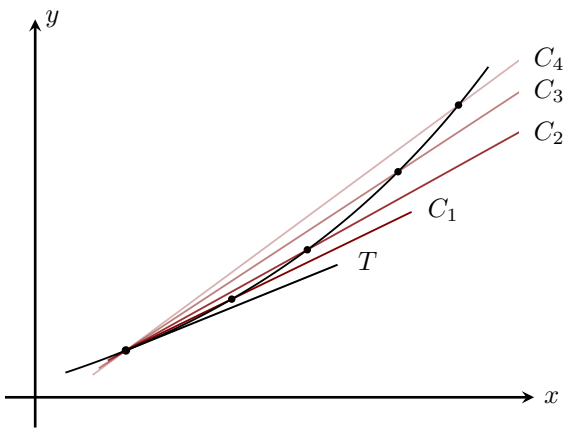

Рис. 8: Касательная как предельное положение секущей.

Секущей графика называется прямая, проходящая через две точки графика. Пусть $f(x)$ — некоторая функция. Секущая, которая будет пересекать график в точках, соответствующих значениям аргумента x_0 и $x_0 + \Delta x$, является единственной и имеет уравнение:

$$y = f(x_0) + \frac{\Delta y}{\Delta x}(x - x_0),$$

где $\Delta y = f(x_0 + \Delta x) - f(x_0)$.

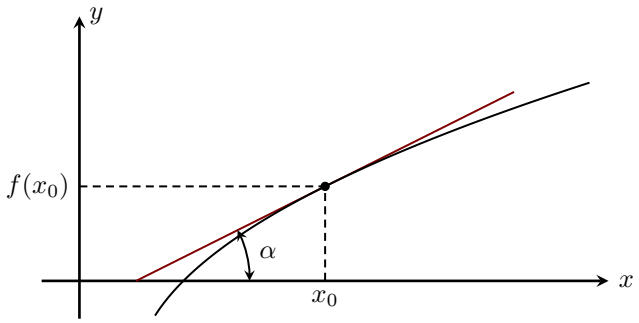

Рис. 9: Геометрический смысл производной

Если теперь при зафиксированном x_0 перейти к пределу $\Delta x \to 0$, уравнение секущей перейдет в уравнение касательной. Действительно, уравнение касательной имеет вид:

$$y = f(x_0) + f'(x_0)(x - x_0) = f(x_0) + \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x}(x - x_0).$$

В качестве наглядного примера можно рассмотреть кусочно-гладкую непрерывную функцию. Пусть, например, в точке $x = 0$ производная испытывает разрыв. Это соответствует тому, что угол наклона касательной тоже испытывает скачок при переходе через эту точку. Появляется «излом». Другими словами, гладкие функции не могут иметь изломов, поскольку производная непрерывна. Это очень хорошо соответствует интуитивным представлениям о гладких функциях.

4. Производная сложной функции

Пусть имеются две функции $f(x)$ и $h(x)$, причем область значений f принадлежит области определения функции h. Тогда функция $h(f(x))$ является применением одной функции к результату другой и называется сложной функцией.

Например, $f(x) = 1 + x$, а $h(x) = \ln(x)$, тогда сложной функцией будет $g(x) = h(f(x)) = \ln(1 + x)$.

Дифференциалом df функции $f(x)$ в точке x_0 называется линейная часть её приращения. Существует связь между дифференциалом функции и её производной:

$$df = f'(x_0)dx, \qquad dx = \Delta x.$$

Именно это выражение послужило основанием для использования знака дифференциала в обозначении Лейбница для производной:

$$f'(x_0) = \frac{df}{dx}(x_0).$$

Следует особо отметить, что выражение $\frac{df}{dx}$ следует трактовать как единый символ, а не частное двух величин.

Теперь можно сделать набросок доказательства так называемого Chain Rule, известного для русскоязычной аудитории как правило для вычисления производной сложной функции. Пусть $z = g(y)$, $y = h(x)$, тогда:

$$\frac{dz}{dx} = \lim_{\delta x \to 0}\left(\frac{\delta z}{\delta x}\right) = \lim_{\delta x \to 0}\left(\frac{\delta z}{\delta y}\frac{\delta y}{\delta x}\right)$$

Строго говоря, приращение δy может равняться нулю даже, если $\delta x \neq 0$. Полное доказательство должно включать в себя громоздкое обсуждение этого момента. Поэтому было решено произвести только набросок доказательства.

$$= \lim_{\delta x \to 0}\left(\frac{\delta z}{\delta y}\right)\lim_{\delta x \to 0}\left(\frac{\delta y}{\delta x}\right) =$$

Поскольку функция $y = h(x)$ является непрерывной, при $\delta x \to 0$ также выполняется $\delta y \to 0$.

$$= \lim_{\delta y \to 0}\left(\frac{\delta z}{\delta y}\right)\lim_{\delta x \to 0}\left(\frac{\delta y}{\delta x}\right) = \frac{dz}{dy}\frac{dy}{dx}.$$

Таким образом получается требуемое выражение для производной сложной функции:

$$\frac{dg(h(x))}{dx} = \frac{dg(y)}{dy}\frac{dh(x)}{dx}.$$

5. Экстремум

Точка x_0 называется локальным минимумом функции $f(x)$, если существует такая окрестность этой точки U_{x_0}, что для всех точек $x \in U_{x_0}$ выполняется $f(x) \geq f(x_0)$. Аналогично вводится понятие максимума функции $f(x)$. Точка локального минимума или локального максимума называется точкой локального экстремума функции $f(x)$.

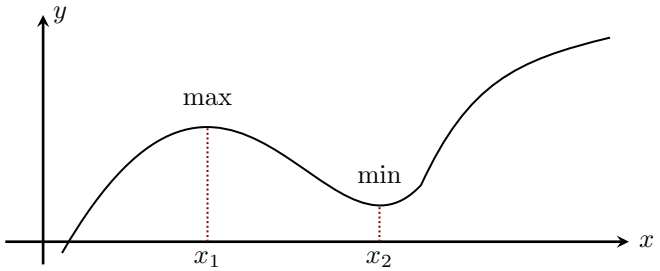

Рис. 10: Функция имеет локальный минимум и локальный максимум

Точка x_0 называется глобальным минимумом функции $f(x)$, если для всех точек $x \in D$ выполняется $f(x) \geq f(x_0)$. Аналогично вводится понятие глобального максимума функции.

Пусть некоторая функция дифференцируема в точке x_0 и известно, что ее производная в этой точке отлична от нуля. В таком случае в достаточно малой окрестности функция будет вести себя линейно с ненулевым коэффициентом наклона, то есть в этой окрестности не может быть никакого экстремума. Другими словами, если функция дифференцируема в точке x_0, то необходимым условием экстремума является $f'(x_0) = 0$.

При этом данное условие, очевидно не является достаточным. В качестве примера можно рассмотреть функцию $y = x^3$. Также, если производная не существует в точке x_0, там все равно может оказаться экстремум функции. В качестве примера можно рассмотреть $y = |x|$.

6. Выпуклость функции

Монотонность функции и выпуклость функции По знаку первой производной гладкой функции на каком-нибудь интервале можно сделать вывод относительно того, возрастает или убывает функция:

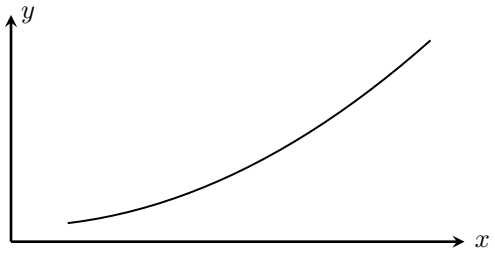

Рис. 11: Выпуклая функция

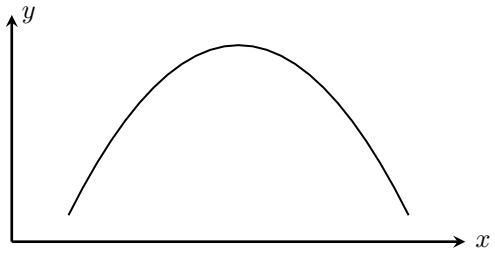

Рис. 12: Вогнутая функция

1. $f'(x) \geq 0$ — функция возрастает,
2. $f'(x) > 0$ — функция строго возрастает,
3. $f'(x) \leq 0$ — функция убывает,
4. $f'(x) < 0$ — функция строго убывает.

Функции, первая производная которых $f'(x)$ ведет себя монотонно, обладают свойством выпуклости ($f'(x)$ монотонно растет, касательная увеличивает свой коэффициент с ростом x) или вогнутости ($f'(x)$ монотонно убывает, касательная уменьшает свой коэффициент).

Учитывая все вышесказанное, если у функции существует вторая производная:

1. $f''(x) \geq 0$ — функция $f(x)$ выпукла,
2. $f''(x) > 0$ — функция $f(x)$ строго выпукла,
3. $f''(x) \leq 0$ — функция $f(x)$ вогнута,
4. $f''(x) < 0$ — функция $f(x)$ строго вогнута.

Достаточное условие экстремума Пусть выполнено необходимое условие экстремума, то есть в некоторой точке x_0 значение $f'(x_0) = 0$. Если в таком случае

1. $f''(x) > 0$ — функция будет строго выпукла и реализуется строгий минимум.
2. $f''(x) < 0$ — функция будет строго вогнута и реализуется строгий максимум.

Таким образом, если функция является строго выпуклой или строго вогнутой, необходимое условие экстремума будет для нее также и достаточным.

Более общее определение выпуклости Вещественнозначная функция, определённая на некотором интервале, выпукла, если для любых двух значений аргумента x, y и для любого числа $t \in [0, 1]$ выполняется

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y).$$

То есть, если соединить две точки на графике отрезком, он окажется выше графика функции $f(x)$.

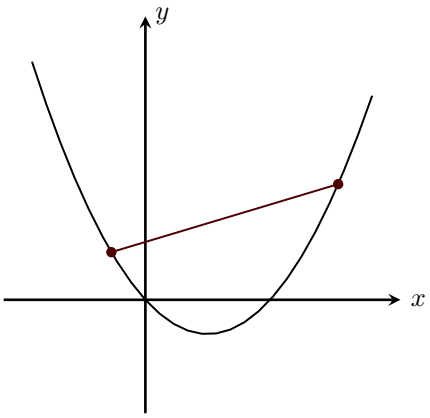

Рис. 13: К определению выпуклости

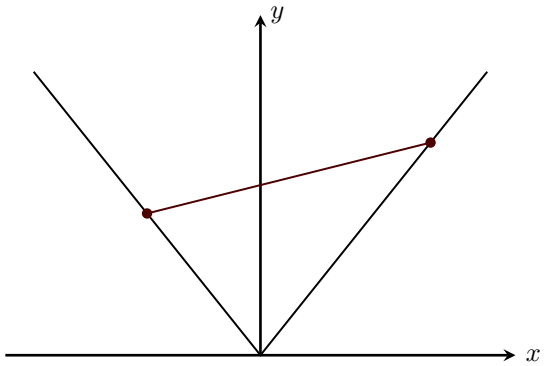

Рис. 14: Функция $|x|$ выпукла

Такое определение подходит и для функций, производная которых не определена в некоторых точках. Например, используя это определение, достаточно просто показать, что $|x|$ — выпуклая функция.

Если функция на каком-либо интервале вогнутая или выпуклая, то на этом интервале не может быть более одного минимума, как не может быть более одного максимума.

Знакомство с линейной алгеброй

1. Признаковое описание

В машинном обучении широко используется понятие признака. Признаком называется отображение из множества объектов в множество допустимых значений этого признака. Если задано множество объектов и некоторый набор признаков, для каждого объекта можно построить его признаковое описание — вектор, составленный из значений этого набора признаков на данном объекте.

Пусть, например, для каждого магазина торговой сети требуется предсказать прибыль в следующем месяце. Эта задача анализа данных имеет огромную практическую ценность. Действительно, если будет выяснено, что прибыль некоторого магазина упадет, можно будет заблаговременно принять меры по предотвращению этого. В этой задаче множеством объектов является множество магазинов. В качестве признаков разумно выбрать следующие:

- Прибыль магазина в каждом из 4-ех последних месяцев (4 признака)
- Планируемое число акций для каждой из трех основных категорий (3 признака)
- Географические координаты магазина: широта и долгота (2 признака)
- Число дней, когда магазин будет открыт в ближайшем месяце. (1 признак)

Признаковым описанием магазина будет вектор, в котором находятся значения данных 10 признаков для данного конкретного магазина.

Если необходимо работать с признаковыми описаниями сразу нескольких объектов, удобно ввести двумерную структуру данных — матрицу, в которой каждая строка соответствует одному объекту, а каждый столбец — признаку. Работать с векторами и матрицами позволяет аппарат линейной алгебры.

2. Векторное пространство

Векторное пространство V представляет собой набор элементов, называемых векторами, для которых определены операции сложения друг с другом и умножения на скаляр, причем эти операции замкнуты и подчинены восьми аксиомам:

1. Коммутативность сложения
2. Ассоциативность сложения
3. Существование нейтрального элемента относительно сложения
4. Существование для каждого вектора x противоположенного вектора $-x$
5. Ассоциативность умножения на скаляр
6. Унитарность: умножение на единичный скаляр не меняет вектор
7. Дистрибутивность умножения на вектор относительно сложения скаляров
8. Дистрибутивность умножения на скаляр относительно сложения векторов

3. Линейная независимость

Линейная зависимость является одним из основополагающих понятий линейной алгебры. Конечный набор элементов векторного пространства называется линейно зависимым, если существует нетривиальная линейная комбинация элементов из этого набора, равная нулевому элементу. Линейная комбинация называется тривиальной, если все коэффициенты в ней равны нулю.

Оказывается, что конечный набор элементов векторного пространства линейно зависим тогда и только тогда, когда один из элементов этого набора может быть выражен через оставшиеся.

С помощью понятия линейной независимости вводится понятие размерности векторного пространства. А именно: размерностью $dimV$ векторного пространства V называется максимальное число линейно независимых векторов в нем.

Пусть дан некоторый набор объектов X и набор $f_j : X \to \mathbb{R}$ вещественнозначных признаков. В этом случае набор векторов признаков может быть линейно зависим. Например, признаки вес товара на первом складе, вес товара на втором складе и вес товара на обоих складах линейно зависимы. Другой случай — два вещественнозначных признака отличаются множителем, например являются одной и той же величиной в разных единицах измерения. В обоих случаях наблюдается избыточность информации.

Такая избыточность приводит к дополнительным затратам ресурсов. Более того, линейная зависимость векторов признаков приводит к возникновению проблем при обучении линейной регрессионной модели — об этом пойдет речь в следующем курсе.

Чтобы проверить, являются ли система векторов линейно зависимой, можно составить из них матрицу и вычислить ее ранг. Об этом будет рассказано далее.

4. Норма и скалярное произведение векторов

Нормированные пространства

Для обобщения понятия длины вектора используется понятие нормы. Функция $\|\cdot\| : V \to \mathbb{R}$ называется нормой в векторном пространстве V, если для нее выполняются аксиомы нормы:

1. $\|x\| = 0 \iff x = 0$ (Нулевую норму имеет только нулевой вектор)
2. $\forall x, y \in L : \|x + y\| \leq \|x\| + \|y\|$ (Неравенство треугольника)
3. $\forall \alpha \in \mathbb{R}\ \forall x \in V : \|\alpha x\| = |\alpha|\|x\|$ (Условие однородности)

Пространство с введенной на нем нормой называют нормированным пространством. Обычно используется Евклидова норма $\|x\|_2$, другой пример нормы — Манхэттенская норма $\|x\|_1$:

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}, \qquad \|x\|_1 = \sum_{i=1}^{n} |x_i|.$$

Метрические пространства

Понятие расстояние обобщается с помощью понятия метрики. Пусть X — некоторое множество, а числовая функция $d : X \times X \to \mathbb{R}$, которая называется метрикой, удовлетворяет следующим условиям:

1. $d(x,\ y) = 0 \Leftrightarrow x = y$ (аксиома тождества).
2. $d(x,\ y) = d(y,\ x)$ (аксиома симметрии).
3. $d(x,\ z) \leqslant d(x,\ y) + d(y,\ z)$ (неравенство треугольника).

Любое нормированное пространство можно превратить в метрическое, определив функцию расстояния $d(x,\ y) = \|y - x\|$. Например, Евклидова и Манхэттенская метрики имеют вид:

$$\rho_2(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}, \qquad \rho_1(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

Название «манхэттенская метрика» связано с уличной планировкой Манхэттена.

Скалярное произведение

Скалярным произведением называется функция $\langle x, y\rangle : V \times V \to \mathbb{R}$, удовлетворяющая следующим условиям:

1. $\langle \alpha x_1 + \beta x_2, y\rangle = \alpha\langle x_1, y\rangle + \beta\langle x_2, y\rangle; \quad \forall \alpha, \beta \in \mathbb{R}\ \forall x_1, x_2, y \in V$
2. $\langle y, x\rangle = \langle x, y\rangle; \quad \forall x, y \in V$
3. $\langle x, x\rangle > 0; \quad \forall x \in V \setminus \{0\}; \quad \langle 0, 0\rangle = 0;$

В современном аксиоматическом подходе уже на основе понятия скалярного произведения векторов вводятся следующие производные понятия:

1. Длина вектора $\|x\| = \sqrt{\langle \mathbf{x}, \mathbf{x}\rangle}$
2. Угол между векторами $\alpha = \arccos \frac{\langle \mathbf{a},\mathbf{b}\rangle}{\sqrt{\langle \mathbf{a},\mathbf{a}\rangle \langle \mathbf{b},\mathbf{b}\rangle}}$.

В случае Евклидового пространства скалярное произведение задается формулой $\langle \mathbf{a}, \mathbf{b}\rangle = \sum_{i=1}^{n} a_i b_i$ и можно убедиться, что такое определение согласуется с введенной ранее Евклидовой нормой:

$$\|x\| = \sqrt{\langle \mathbf{x}, \mathbf{x}\rangle} = \sqrt{\sum_{i=1}^{n} x_i^2} = \|x\|_2$$

Матрицы и основные матричные операции

1. Определение матриц

Матрицей размера $n \times m$ называется прямоугольная таблица специального вида, состоящая из n строк и m столбцов, заполненная числами. Матрица обычно обозначаются заглавными буквами латинского алфавита, например A. Элементы матрицы A обозначаются a_{ij}, где i и j — номер строки и столбца, где расположен этот элемент, соответственно. Пространство матриц $n \times m$ обозначается $\mathbb{R}^{n \times m}$.

Матрицы можно использовать для работы с системами линейных алгебраических уравнений. Например, в задачах линейной классификации решается система линейных алгебраических уравнений с матрицей объект-признак относительно вектора неизвестных параметров. При этом получившаяся система уравнений может как иметь бесконечное число решении, так и не иметь решений вовсе. Подробнее эти вопросы будут обсуждаться в курсе по машинному обучению.

Для матриц размера $m \times n$ результат операции умножения на вектор-столбец размера n (т.е. на матрицу размера $n \times 1$) есть новый вектор-столбец размера m:

$$Aw = \begin{pmatrix} \sum_{i=1}^{n} a_{1i} w_i \\ \sum_{i=1}^{n} a_{2i} w_i \\ \dots \\ \sum_{i=1}^{n} a_{mi} w_i \end{pmatrix}$$

Другими словами, можно сказать, что матрица задает линейное отображение.

Системы линейных уравнений лаконично записываются с помощью матриц, например:

$$\begin{cases} 12w_1 + 7w_2 + 21w_3 + 31w_4 + 11w_5 = 1 \\ 45w_1 - 2w_2 + 14w_3 + 27w_4 + 19w_5 = 0 \\ -3w_1 + 15w_2 + 36w_3 + 71w_4 + 21w_5 = 0 \\ 4w_1 - 13w_2 + 55w_3 + 34w_4 + 15w_5 = 1 \end{cases}$$

в матричной записи имеет вид:

$$\begin{pmatrix} 12 & 7 & 21 & 31 & 11 \\ 45 & -2 & 14 & 27 & 19 \\ -3 & 15 & 36 & 71 & 21 \\ 4 & -13 & 55 & 34 & 15 \end{pmatrix} \cdot \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

2. Матричные операции

Произведение матриц

Ранее уже было сказано, как производится умножение матрицы размера $m \times n$ на вектор-столбец длины n:

$$A \cdot w = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \ddots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_n \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} a_{1i} w_i \\ \sum_{i=1}^{n} a_{2i} w_i \\ \dots \\ \sum_{i=1}^{n} a_{mi} w_i \end{pmatrix}.$$

При этом следует отметить, что такая операция возможна только тогда, когда число столбцов матрицы совпадает с длиной вектора. Иначе операция не определена. Можно определить более общую операцию умножения матриц следующим образом.

Пусть $A \in \mathbb{R}^{m \times n}$ и $B \in \mathbb{R}^{n \times k}$ — две матрицы соответствующих размеров (число столбцов в A совпадает с числом строк во B). Тогда произведением матрицы A на матрицу B называется матрица C размера $m \times k$:

$$C = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1k} \\ c_{21} & c_{22} & \cdots & c_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mk} \end{pmatrix}, \qquad c_{ij} = \sum_{p=1}^{n} a_{ip} b_{pj}.$$

Пример: произведение матриц

Заданы две матрицы согласованного размера:

$$A = \begin{pmatrix} 1 & 2 \\ 0 & 1 \\ 10 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 2 \end{pmatrix},$$

тогда их произведение:

$$A \cdot B = \begin{pmatrix} 1 \cdot 1 + 2 \cdot 0 & 1 \cdot 0 + 2 \cdot 0 & 1 \cdot 0 + 2 \cdot 2 \\ 0 \cdot 1 + 1 \cdot 0 & 0 \cdot 0 + 1 \cdot 0 & 0 \cdot 0 + 1 \cdot 2 \\ 10 \cdot 1 + 0 \cdot 0 & 10 \cdot 0 + 0 \cdot 0 & 10 \cdot 0 + 0 \cdot 2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 4 \\ 0 & 0 & 2 \\ 10 & 0 & 0 \end{pmatrix}$$

Произведение матриц и линейные отображения

Ранее было сказано, что матрицы задают линейные отображения. Матрица $A \in \mathbb{R}^{m \times n}$ размера $m \times n$ будет задавать отображение из пространства векторов длины n в пространство векторов размера m. Аналогично матрица $B \in \mathbb{R}^{n \times k}$ задает отображение из пространства векторов длины k в пространство векторов длины n.

Но тогда можно построить отображение из пространства векторов длины n в пространство векторов длины k последовательным применением отображений, задаваемых матрицами A и B:

$$y_j = \sum_{j=1}^{n} a_{ij} \left(\sum_{p=1}^{k} b_{jp} w_p \right) = \sum_{p=1}^{k} \left(\sum_{j=1}^{n} a_{ij} b_{jp} \right) w_p = \sum_{p=1}^{k} c_{ip} w_p.$$

Матрица получившегося преобразования в точности равна произведению матриц A и B:

$$c_{ip} = \sum_{j=1}^{n} a_{ij} b_{jp}.$$

Определённое таким образом произведение матриц оказывается совершенно естественным.

Например, при решении системы линейных уравнений $Ax = b$ (например при решении задачи линейной классификации) на x могут дополнительно быть наложены определенные ограничения. Если такие ограничения могут быть введены путем замены $x = Bz$, то вектор z может быть найдет из системы уравнений:

$$ABz = b,$$

матрица которой в точности равна произведению матриц A и B.

Сложение и умножение на число

Для матриц также определены операции сложения и умножения на число. **Складывать можно только матрицы одинакового размера.** Сложение матриц происходит поэлементно, то есть $c_{ij} = a_{ij} + b_{ij}$, если $C = A + B$, где A и B — матрицы одного размера. Умножение матрицы на число заключается в умножении каждого элемента матрицы на это число, то есть $d_{ij} = \lambda a_{ij}$, если $D = \lambda A$.

Другими словами, матрицы одного размера образуют векторное пространство.

Транспонирование матриц

Для матриц также определяют операцию транспонирования. Матрица $B = A^T$, полученная транспонированием матрицы A размера $m \times n$, будет иметь размер $n \times m$, а её элементы будут связаны с элементами исходной матрицы следующим образом: $b_{ij} = a_{ji}$. Говоря нестрого, столбцы исходной матрицы стали строками новой матрицы, а строки — столбцами.

Пример: транспонирование матриц

$$A = \begin{pmatrix} 1 & 4 & 1 \\ 2 & 3 & 3 \\ 2 & 1 & 0 \\ 5 & 1 & 1 \end{pmatrix}, \qquad A^T = \begin{pmatrix} 1 & 2 & 2 & 5 \\ 4 & 3 & 1 & 1 \\ 1 & 3 & 0 & 1 \end{pmatrix}.$$

3. Ранг и определитель

Определитель матрицы 2×2: геометрический смысл

Важным понятием линейной алгебры является определитель (детерминант) матрицы. Он имеет множество приложений, в том числе и геометрических. Например, с помощью него можно вычислить площадь построенного на двух векторах параллелограмма на плоскости.

Для этого каждый из векторов следует записать как отдельный столбец матрицы A размера 2×2. Тогда преобразование, задаваемое матрицей A, будет отображать единичный квадрат в требуемый параллелограмм. Определитель $\det A$ с точностью до знака будет равен площади параллелограмма:

$$\det A = a_{11}a_{22} - a_{12}a_{21}, \qquad S = |\det A|.$$

Таким образом, абсолютное значение определителя отражает коэффициент изменения площади при преобразовании, а знак показывает, выполняет ли преобразование A отражение. Вообще говоря, свойства преобразования можно изучать на основе того, в какой параллелограмм оно переводит единичный квадрат.

Пример: нахождение площади параллелограмма

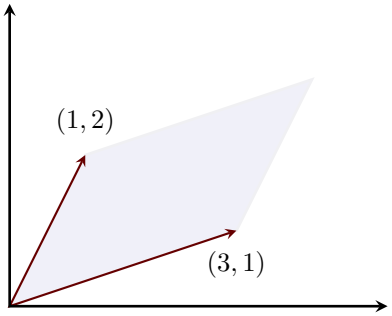

Рис. 1: Площадь параллелограмма.

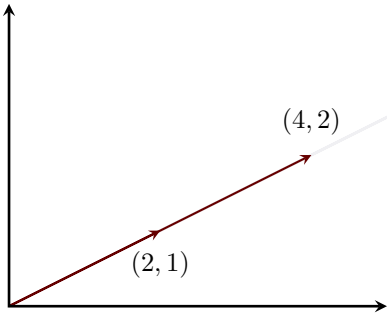

Рис. 2: Вырожденный случай.

Для того, чтобы вычислить площадь параллелограмма, построенного на векторах $(1, 2)^T$ и $(3, 1)^T$, вычислим определитель матрицы, составленной из этих векторов:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 1 \end{pmatrix}, \qquad S = |\det A| = |3 \cdot 2 - 1 \cdot 1| = 5.$$

Определитель матрицы

Понятие определителя матрицы имеет смысл только для квадратных матриц. Дать определение определителю на случай пространств больших размерностей можно несколькими способами: существуют определение через перестановки, определение через формулу разложения по строке и аксиоматическое определение. Геометрический смысл определителя некоторой матрицы — ориентированный объем многомерного параллелепипеда, построенного на векторах, которые являются столбцами этой матрицы.

Можно дать определение используя следующую рекурсивную формулу (разложение по строке):

$$\det A = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} \bar{M}_j^1,$$

где $\bar{M}_j^1$ — определитель матрицы, полученной из исходной вычеркиванием 1-строки и j-го столбца.

Свойства определителя

Определитель широко используется в линейной алгебре, так как обладает целым рядом важных свойств:

- Определитель матрицы, содержащей линейно зависимые строки или столбцы, равен нулю.
- Определитель не меняется при транспонировании.
- Если $A, B \in \mathbb{R}^{n \times n}$ — квадратные матрицы одного размера, то $\det(AB) = \det A \cdot \det B$.

Ранг матрицы

Ранг матрицы $\operatorname{rg} A$ — другое важное понятие из линейной алгебры. Сначала дают два определения для ранга матрицы:

- **Строчный ранг:** называется максимальное число линейно независимых строк.
- **Столбцовый ранг:** называется максимальное число линейно независимых столбцов.

Фундаментальная теорема «о ранге матрицы» говорит, что строчный ранг равен столбцовому, и поэтому говорят просто о ранге матрицы.

4. Системы линейных уравнений (СЛАУ)

Систему из m линейных уравнений относительно n неизвестных x_1,..., x_n можно записать в общем виде:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

Но удобно использовать матричную форму СЛАУ. Система линейных алгебраических уравнений может быть представлена в матричной форме как:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix},$$

или более кратко $Ax = b$, где A — это матрица коэффициентов, b — столбец свободных членов, а x — вектор из неизвестных. Причем возможны три случая:

1. Система вовсе не имеет решения. Такая система называется несовместной.
2. Система имеет единственное решение.
3. Система имеет бесконечно много решений.
   Это связано с тем, что если система имеет два решения $\vec{x}_1$ и $\vec{x}_2$, то любой вектор вида $t\vec{x}_1 + (1-t)\vec{x}_2$, где $t \in \mathbb{R}$, также будет решением.

Теорема Кронекера-Капелли дает критерий совместности системы. Система линейных алгебраических уравнений $Ax = b$ совместна тогда и только тогда, когда ранг её основной матрицы A равен рангу её расширенной матрицы $(A|b)$, причём система имеет единственное решение, если ранг равен числу неизвестных, и бесконечное множество решений, если ранг меньше числа неизвестных. Расширенная матрица получается из матрицы A приписыванием к ней справа столбца b.

Решение системы уравнений с квадратной матрицей можно записать с привлечением так называемой обратной матрицы. Матрица называется обратной к матрице A и обозначается A^{-1}, если выполнено:

$$AA^{-1} = A^{-1}A = I,$$

где I — единичная матрица нужного размера. **Обращение возможно только в случае квадратных невырожденных матриц.** Действительно, если предположить, что у вырожденной матрицы A ($\det A = 0$) существовала бы обратная матрица, легко можно прийти к противоречию:

$$1 = \det I = \det(A^{-1}A) = \det A^{-1} \det A = \det A^{-1} \cdot 0 = 0.$$

Решение уравнения $Ax = b$, если матрица A — невырожденная, единственно и запишется в виде:

$$x = A^{-1}b.$$

Задача вычисления обратной матрицы и решения СЛАУ, строго говоря, сравнимы по сложности.

5. Типы матриц

Неоднократно было сказано, что матрица задаёт линейное отображение из одного векторного пространства в другое. Особый интерес представляет случай, когда строится отображение из некоторого линейного пространства в его само же. Такой класс линейных отображений называется линейными преобразованиями. Матрицы линейных преобразований всегда квадратные.

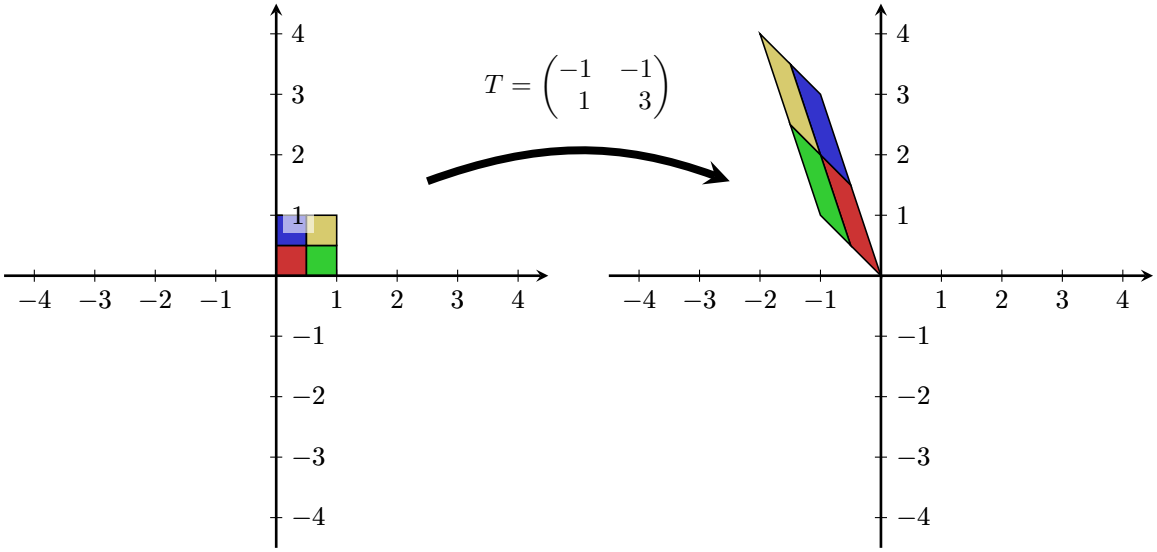

Рис. 3: Действие линейного преобразования, задаваемого матрицей T, на единичный квадрат.

Квадратная матрица называется диагональной в том случае, если все её элементы, стоящие вне главной диагонали, равны нулю. Частным случаем диагональной матрицы является единичная матрица I, на главной диагонали которой стоят единицы. Если матрица преобразования диагональная, то она задает растяжение или сжатие по осям системы координат.

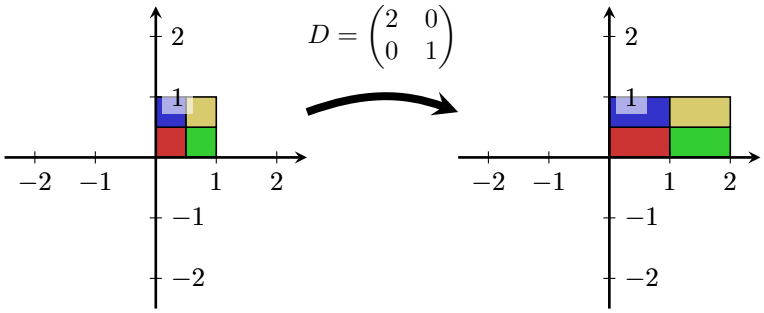

Рис. 4: Действие линейного преобразования, задаваемого диагональной матрицей D.

Важный класс квадратных матриц — ортогональные матрицы. Матрица называется ортогональной, если:

$$A^T A = AA^T = I.$$

Непосредственно из определения следуют следующие важные свойства:

- Ортогональная матрица обратима, причем $A^{-1} = A^T$.
- Ортогональные преобразования сохраняют скалярное произведение:

$$\langle Ax, Az \rangle = (Ax)^T (Az) = x^T A^T Az = x^T z = \langle x, z \rangle.$$

- Ортогональные преобразования сохраняют длины векторов $\|Ax\| = \|x\|$ и углы между векторами.

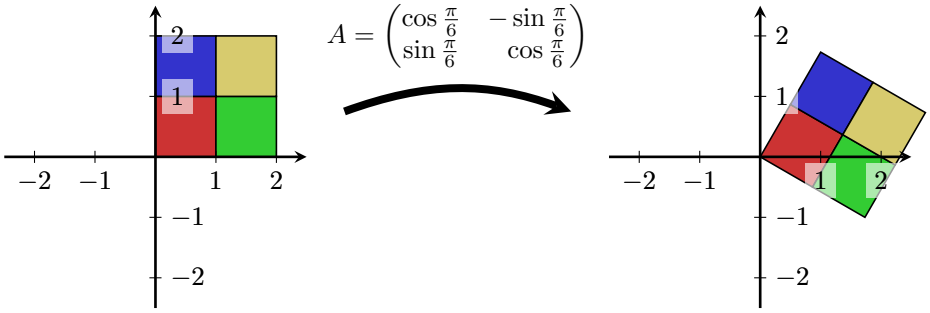

Рис. 5: Действие линейного преобразования, задаваемого ортогональной матрицей A.

В случае евклидовой плоскости всякое ортогональное преобразование является поворотом или поворотом с инверсией, и его матрица в любом ортонормированном базисе имеет вид

$$\begin{pmatrix} \cos\varphi & \sin\varphi \\ -\sin\varphi & \cos\varphi \end{pmatrix} \quad \text{или} \quad \begin{pmatrix} \cos\varphi & -\sin\varphi \\ \sin\varphi & \cos\varphi \end{pmatrix}.$$

Другой важнейший класс матриц — симметричные. Матрица A называется симметричной, если $A = A^T$.

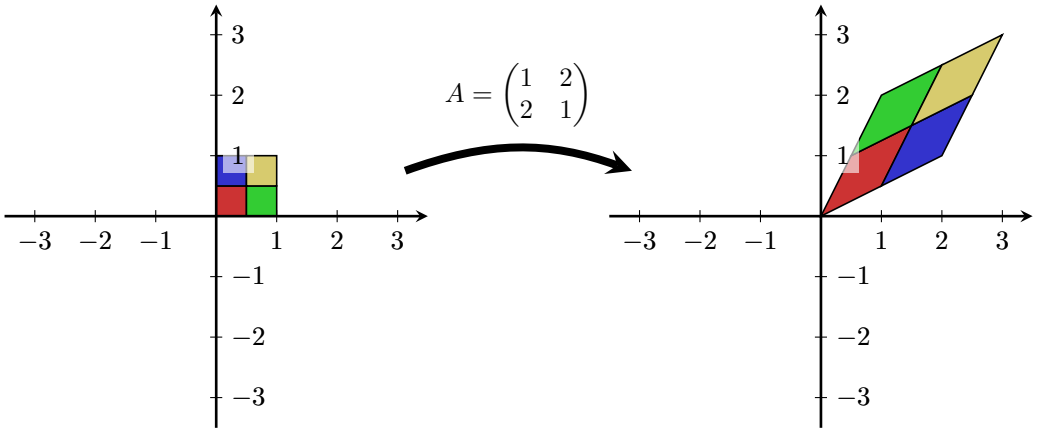

Рис. 6: Действие линейного преобразования, задаваемого симметричной матрицей A.

Важным результатом линейной алгебры является теорема о приведении симметричных матриц к диагональному виду с помощью ортогонального преобразования. А именно всякую симметричную матрицу $A = A^T$ можно записать как $A = QDQ^T$, где Q — некоторая ортогональная матрица, а D — диагональная матрица.

6. Понятие собственного вектора

Важной характеристикой матрицы, а также линейного преобразования, заданного этой матрицей, является спектр — набор собственных векторов и соответствующих собственных значений.

Собственным вектором линейного преобразования A называется такой ненулевой вектор $x \in V$, что для некоторого $\lambda \in \mathbb{R}$ выполняется $Ax = \lambda x$.

Линейное преобразование может как не иметь собственных векторов вообще, например поворот в двумерном пространстве (кроме нескольких исключительных случаев), или иметь n собственных векторов с различными собственными значениями. Вопросы существования собственных векторов преобразования разбираются в курсе линейной алгебры.

Понятие собственного вектора используется в Методе Главных Компонент, который предназначен для уменьшения размерности данных с потерей наименьшего количества информации.

Функции многих переменных

1. Частные производные и градиент

Понятие частной производной

Частная производная — это одно из обобщений понятия производной на случай функции нескольких переменных.

Частная производная функции $f(x, y)$ по x определяется как производная по x, взятая в смысле функции одной переменной, при условии постоянства оставшейся переменной y. Таким образом:

$$f'_x(x,y) = \lim_{h\to 0} \frac{f(x+h,y) - f(x,y)}{h}, \qquad f'_y(x,y) = \lim_{h\to 0} \frac{f(x,y+h) - f(x,y)}{h}.$$

Геометрический смысл частной производной

Пусть дана некоторая функция двух переменных $f : \mathbb{R}^2 \to \mathbb{R}$. Она, вообще говоря, определяет некоторую поверхность $z = f(x, y)$ в трехмерном пространстве.

Если в некоторой точке (x_0, y_0) функция дифференцируема как функция многих переменных, то в этой точке можно рассмотреть плоскость касательную к рассматриваемой поверхности.

Эта касательная плоскость пересекает координатные плоскости $x0z$ и $y0z$ по прямым, тангенсы угла между которыми и соответствующими координатными осями равны значениям частных производных в точке (x_0, y_0).

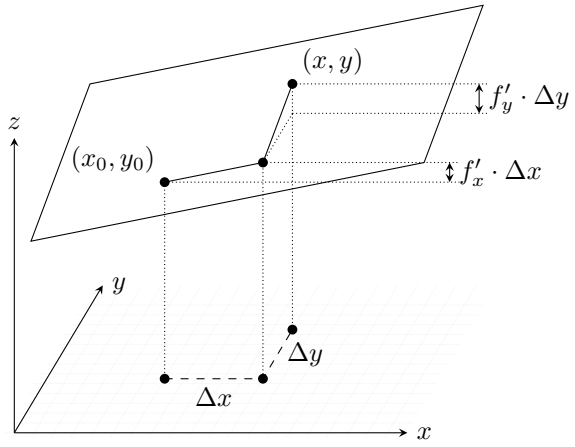

Рис. 1: Геометрический смысл частных производных.

Таким образом, график функции $f(x, y)$ в окрестности точки можно приблизить касательной плоскостью:

$$f(x_0 + \Delta x, y_0 + \Delta y) \approx f(x_0, y_0) + f'_x(x_0, y_0)\Delta x + f'_y(x_0, y_0)\Delta y.$$

Градиент

Если $f(x_1, \ldots, x_n)$ — функция n переменных $x_1, ..., x_n$, то n-мерный вектор из частных производных:

$$\operatorname{grad} f = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right)$$

называется градиентом.

Линией уровня функции называется множество точек, в которых функция принимает одно и то же фиксированное значение. Оказывается, что градиент перпендикулярен линии уровня. Более подробное обсуждение этого факта будет произведено позднее.

2. Градиент в задачах оптимизации

Задачей оптимизации называется задача по нахождению экстремума функции, например минимума:

$$f(x_1, ..., x_n) \to \min .$$

Такая задача часто встречается в приложениях, например при выборе оптимальных параметров рекламной компании, а также в задачах классификации.

Если функция дифференцируема, то найти точки, подозрительные на экстремум, можно с помощью необходимого условия экстремума: все частные производные должны равняться нулю, а значит вектор градиента — нулевому вектору.

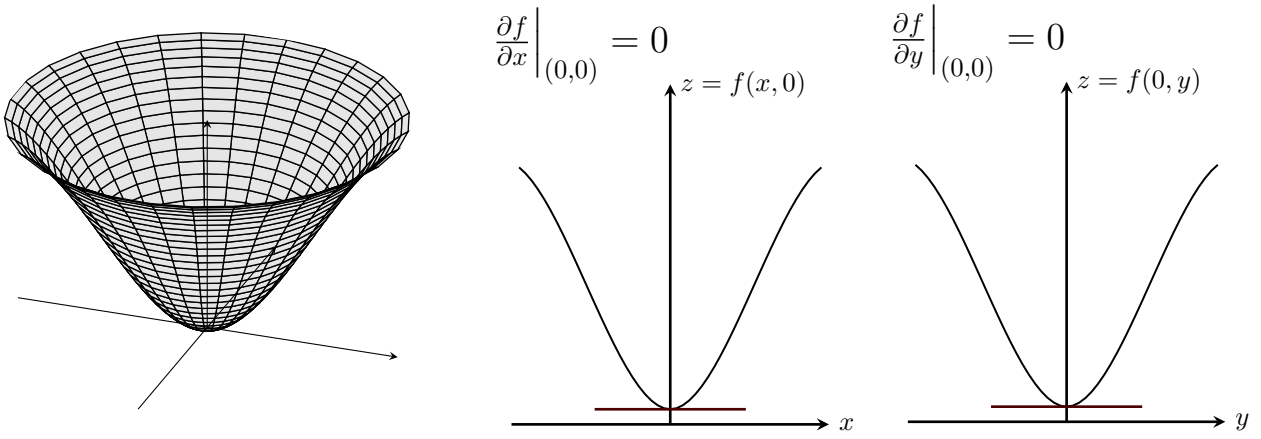

Рис. 2: Функция двух переменных достигает минимума в начале координат.

Но не всегда задачу можно решать аналитически. В таком случае используется численная оптимизация.

Наиболее простым в реализации из всех методов численной оптимизации является метод градиентного спуска. Это итерационный метод. Решение задачи начинается с выбора начального приближения $\vec{x}^{[0]}$. После вычисляется приблизительное значение $\vec{x}^1$, а затем $\vec{x}^2$ и так далее, согласно итерационной формуле:

$$\vec{x}^{[j+1]} = \vec{x}^{[j]} - \gamma^{[j]} \nabla F(\vec{x}^{[j]}), \qquad \text{где } \gamma^{[j]} \text{ — шаг градиентного спуска.}$$

Основная идея метода заключается в том, чтобы идти в направлении наискорейшего спуска, а это направление задаётся антиградиентом $-\nabla F$.

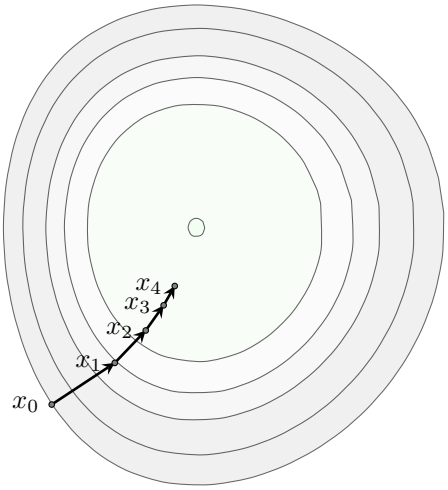

Рис. 3: Градиентный спуск

Можно привести следующую аналогию: Представьте, что вы приехали в незнакомое место и поселились где-то в низине. Гуляя по недалеким окрестностям, вы заблудились и теперь хотите попасть домой. Если

вы не ориентируетесь в местности, логичным решением будет идти в каждый момент времени в направлении наискорейшего спуска, чтобы вернуться в свой дом.

3. Производная по направлению

Пусть $f(\vec{x}) = f(x_1, x_2, \ldots, x_n)$ — функция n переменных, $\vec{\ell} \in \mathbb{R}^n$, $|\vec{\ell}| = 1$, тогда частной производной в точке x_0 по направлению $\vec{\ell}$ называется

$$\frac{\partial f}{\partial \vec{\ell}}(\vec{x}_0) = \lim_{t \to 0} \frac{f(\vec{x}_0 + t \cdot \vec{\ell}) - f(\vec{x}_0)}{t}.$$

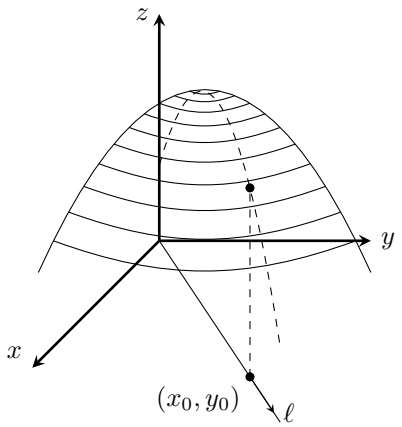

Рис. 4: К определению производной по направлению функции двух аргументов.

Непосредственно из определения становится понятен геометрический смысл производной по направлению. Производная по направлению показывает, насколько быстро функция изменяется при движении вдоль заданного направления. Производная по направлению координатной оси является частной производной по соответствующей координате.

4. Касательная плоскость и линейное приближение

Пусть функция $f(x, y)$ дифференцируема в точке (x_0, y_0), тогда в окрестности этой точки можно записать:

$$f(x_0 + \Delta x, y_0 + \Delta y) \approx f(x_0, y_0) + f'_x(x_0, y_0)\Delta x + f'_y(x_0, y_0)\Delta y.$$

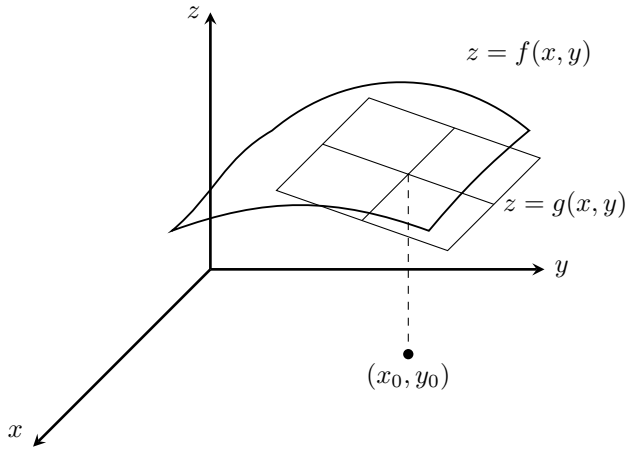

Рис. 5: Касательная плоскость к функции двух переменных.

Это выражение может быть переписано более удобных обозначениях:

$$\Delta f \approx \left\langle \nabla f(x_0, y_0), \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \right\rangle.$$

Это есть не что иное, как уравнение касательной плоскости, записанное через скалярное произведение. Следует отметить, что линейное приближение тем лучше описывает поведение функции, чем ближе к точке (x_0, y_0) находится интересующее значение.

Линейное приближение часто используется для анализа сложных функций, в частности, при построении алгоритмов анализа данных.

5. Направление наискорейшего роста

Если функция $f(x, y)$ дифференцируема в точке (x_0, y_0), то в окрестности этой точки для функции можно воспользоваться линейным приближением:

$$\Delta f \approx \left\langle \nabla f(x_0, y_0), \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} \right\rangle.$$

Пусть $\vec{\ell} \in \mathbb{R}^2$, $|\vec{\ell}| = 1$, тогда приращения можно задать вдоль вектора $\vec{\ell}$:

$$\Delta x = t \cdot \ell_x, \qquad \Delta y = t \cdot \ell_y.$$

Если теперь подставить эти приращения в выражение для линейного приближения:

$$\Delta f \approx \left\langle \nabla f(x_0, y_0), \begin{pmatrix} t \cdot \ell_x \\ t \cdot \ell_y \end{pmatrix} \right\rangle = t \cdot \left\langle \nabla f(x_0, y_0), \vec{\ell} \right\rangle,$$

можно получить связь производной по направлению и градиентом:

$$\frac{\partial f}{\partial \vec{\ell}}(x_0, y_0) = \lim_{t \to 0} \frac{\Delta f}{t} = \left\langle \nabla f(x_0, y_0), \vec{\ell} \right\rangle$$

Таким образом, производная по направлению может быть вычислена как скалярное произведение градиента на соответствующий единичный вектор:

$$\frac{\partial f}{\partial \vec{\ell}}(x_0, y_0) = \left\langle \nabla f(x_0, y_0), \vec{\ell} \right\rangle.$$

Согласно данной формуле, направление максимального роста — это направление задаваемое градиентом. Действительно, скалярное произведение $\left\langle \nabla f(x_0, y_0), \vec{\ell} \right\rangle$ максимально в том случае, когда векторы $\nabla f(x_0, y_0)$ и $\vec{\ell}$ сонаправлены.

Методы оптимизации

1. Оптимизация негладких функций

Проблема оптимизации параметров алгоритма машинного обучения

На самом деле, использовать методы оптимизации функций, требующие существование градиента, получается не всегда. Даже если градиент у интересующей функции существует, часто оказывается, что вычислять его непрактично.

Например, существует проблема оптимизации параметров $\alpha_1, ..., \alpha_N$ некоторого алгоритма машинного обучения. Задача оптимизации состоит в том, чтобы подобрать эти параметры так, чтобы алгоритм давал наилучший результат. В частности, если качество работы алгоритма описывать функцией качества $Q(\alpha_1, ..., \alpha_N)$ от его параметров, задача оптимизации принимает вид:

$$Q(\alpha_1, ..., \alpha_N) \to \max_{\alpha_1,...,\alpha_N}.$$

Вычисление градиента в этом случае или невозможно в принципе, или крайне непрактично.

Проблема локальных минимумов

Другая проблема градиентных методов — проблема локальных минимумов.

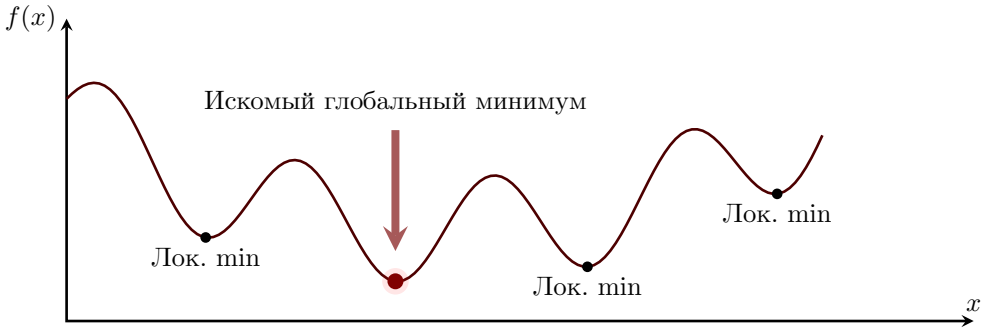

Рис. 1: К вопросу о проблеме локальных минимумов

Метод градиентного спуска, попав на дно локального минимума, где градиент также равен нулю, там и остается. Глобальный минимум так и остается не найденным. Решить эти проблемы позволяют методы случайного поиска. Общая идея этих методов заключается в намеренном введении элемента случайности.

Пример: градиентный спуск без градиента

Если градиент по каким-либо причинам вычислить нельзя, задачу оптимизации можно попытаться решить с помощью модифицированного стохастического алгоритма градиентного спуска.

Пусть решается задача оптимизации $f(\vec{x}) \to \min_{\vec{x}}$, зафиксировано некоторое число d — параметр метода. Используется следующий итерационный процесс:

1. Случайным образом выбирается вектор $\vec{u}$ (случайный вектор $\vec{u}$ равномерно распределен по сфере).
2. Вычисляется значение выражения, которое есть ни что иное, как численная оценка значения производной функции f по направлению $\vec{u}$:
$$\frac{f(\vec{x}) - f(\vec{x} + d\vec{u})}{d}.$$
3. Сдвигаем точку в направлении $\vec{u}$ пропорционально вычисленной на предыдущем шаге величине.

Следует отметить, что ни на каком шаге градиент функции не вычисляется. Более того, направление смещения $\vec{u}$ выбирается случайным образом. Но так как величина смещения зависит от выражения функции в точке $\vec{x} + d\vec{u}$, в среднем происходит смещение по антиградиенту сглаженной функции. Причем, зафиксированное в начале число d, как раз имеет смысл «параметра сглаживания» при нахождении численной оценки производной.

2. Метод имитации отжига

Метод имитации отжига является одним из методов глобальной оптимизации, для работы которого не требуется гладкость функции. Является вариантом метода случайного поиска и известен как алгоритм Метрополиса.

Алгоритм основывается на имитации физического процесса, который происходит при кристаллизации вещества, в том числе при отжиге металлов. Отжиг — вид термической обработки металлов, сплавов, заключающийся в нагреве до определённой температуры и последующем медленном охлаждении до комнатной температуры. Цель отжига — привести систему, которой является образец металла, в состояние с минимальной энергией. Атомы при отжиге могут как попасть в состояние с меньшей энергией, так и в состояние с большей. Причем вероятность попасть из текущего состояния в состоянии с большей энергией уменьшается с температурой. Постепенно температура уменьшается до комнатной и система попадает в состояние с минимальной энергией.

Для задач оптимизации имитация процесса может быть произведена следующим образом. Вводится параметр T, который имеет смысл температуры, и в начальный момент ему устанавливается значение T_0. Набор переменных, по которым происходит оптимизация, будет обозначаться как x.

В качестве начального состояния системы выбирается произвольная точка. Далее запускается итерационный процесс — на каждом шаге из множества соседних состояний случайно выбирается новое x^*. Если значение функции в этой точке меньше, чем значение в текущей точке, то эта точка выбирается в качестве нового состояния системы. В ином случае (т.е. если $f(x^*) > f(x)$) такой переход происходит с вероятностью P, зависящей от температуры T, текущего состояния x и кандидата на новое состояние x^* следующим образом:

$$P = e^{-\frac{f(x^*)-f(x)}{T}}.$$

Благодаря переходам в худшее состояние в методе имитации отжига удалось решить проблему локальных минимумов и не застревать в них. Постепенно, при уменьшении температуры, уменьшается и вероятность переходов в состояния с большим значением функции. Таким образом в конце имитации отжига в качестве x оказывается искомый глобальный минимум.

Оказывается, что успешность этого алгоритма зависит от способа выбора кандидатов, другими словами, того, как именно происходит такой случайный выбор. Такие вопросы как, находит ли этот алгоритм минимум или насколько он эффективен для поиска состояния с меньшим значением функции f (в случае если минимум не может быть найден), представляют отдельный интерес. Стоит также отметить, что практическую ценность могут иметь и алгоритмы, не гарантирующие достижения минимума.

Реализация метода имитации отжига доступна как функция `anneal` (англ. отжиг) в модуле `optimize` библиотеки SciPy.

3. Генетические алгоритмы

Генетические алгоритмы моделируют процесс естественного отбора в ходе эволюции и являются еще одним семейством методов оптимизации. Генетические алгоритмы включают в себя стадии генерации популяции, мутаций, скрещивания и отбора. Порядок стадий завит от конкретного алгоритма.

Алгоритм дифференциальной эволюции

Для оптимизации функции $f(\vec{x})$ вещественных переменных $\vec{x} \in \mathbb{R}^n$ применяется алгоритм дифференциальной эволюции. Популяцией в алгоритме дифференциальной эволюции считается множество векторов из $\mathbb{R}^n$, причем каждая переменная этого пространства соответствует своему признаку. Параметрами этого алгоритма являются: размер популяции N, сила мутации $F \in [0, 2]$ и вероятность мутации P.

В качестве начальной популяции выбирается набор из N случайных векторов. На каждой следующей итерации алгоритм генерирует новое поколение векторов, комбинируя векторы предыдущего поколения. А именно: для каждого вектора x_i из текущего поколения:

Стадия мутации Случайно из популяции выбираются не равные x_i векторы v_1, v_2, v_3. На основе этих векторов генерируется так называемый мутантный вектор:

$$v = v_1 + F \cdot (v_2 - v_3).$$

Стадия скрещивания Над мутантным вектором выполняется операция «скрещивания», в ходе которой каждая координата с вероятность p замещается соответствующей координатой вектора x_i. Получившийся вектор называется пробным.

Стадия отбора Если пробный вектор оказывается лучше исходного x_i (то есть значение исследуемой функции на пробном векторе меньше, чем на исходном), то в новом поколении он занимает его место.

Если сходимость не была достигнута, начинается новая итерация.

Следует также учитывать следующие замечания:

- Для решения задач оптимизации функций дискретных переменных достаточно переопределить только стадию мутации. Остальные шаги алгоритма остаются без изменений.
- Часто, чтобы увеличить эффективность алгоритма, создаются несколько независимых популяций. Для каждой такой популяции формируется свой начальный набор случайных векторов. В таком случае появляется возможность использовать генетические алгоритмы для решения задач глобальной оптимизации.
- Эффективность метода зависит от выбора операторов мутации и скрещивания для каждого конкретного типа задач.

Реализация метода доступна как функция `differentialevolution` в модуле `optimize` библиотеки SciPy.

4. Метод Нелдера-Мида

Метод Нелдера-Мида, или метод деформируемого многогранника, применяется для нахождения решения задачи оптимизации вещественных функций многих переменных

$$f(x) \to \min, \qquad x \in \mathbb{R}^n,$$

причем функция $f(x)$, вообще говоря, не является гладкой и может быть зашумленной. Другой особенностью метода является то, что на каждой итерации вычисляется значение функции $f(x)$ не более чем в трех точках. Это особенно важно в случае сложно вычислимой функции $f(x)$. Метод Нелдера-Мида прост в реализации и полезен на практике, но, с другой стороны, для него не существует теории сходимости — алгоритм может расходиться даже на гладких функциях.

Выпуклым множеством называется такое множество, содержащее вместе с любыми двумя точками соединяющий их отрезок. Выпуклой оболочкой множества называется его минимальное выпуклое надмножество. Симплексом (n-симплексом) называется выпуклая оболочка множества аффинно независимых $(n+1)$ точек (вершин симплекса).

Рис. 2: Отрезок (1-симплекс)

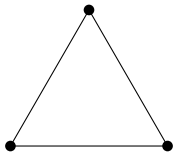

Рис. 3: Треугольник (2-симплекс)

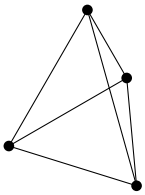

Рис. 4: Тетраэдр (3-симплекс)

Описание алгоритма метода Нелдера-Мида

Основными параметрами метода являются $\alpha > 0$, $\beta > 0$ и $\gamma > 0$ — коэффициенты отражения, сжатия и растяжения соответственно. Пусть необходимо найти безусловный минимум функции n переменных.

1. (Подготовка) Выбирается $n+1$ точка, образующие симплекс n-мерного пространства:

$$x_i = \left(x_i^{(1)}, x_i^{(2)}, ..., x_i^{(n)}\right), \qquad i = 1, ..., n+1.$$

В этих точках вычисляется значение функции $f(x)$:

$$f_1 = f(x_1), f_2 = f(x_2), ..., f_{n+1} = f(x_{n+1}).$$

2. (Сортировка) Среди вершин симплекса $\{x_i\}$ выбираются: точка x_h с наибольшим (из значений на вершинах симплекса) значением функции f_h, точка x_g со следующим по величине значением f_g и точка x_l с наименьшим значением функции f_l.

3. (Центр тяжести) Вычисляется центр тяжести всех точек, за исключением x_h (вычислять значение функции в найденной точке не обязательно):

$$x_c = \frac{1}{n}\sum_{i \neq h} x_i.$$

4. («Отражение») Точка x_h отражается относительно точки x_c с коэффициентом α. В получившейся точке x_r вычисляется значение функции:

$$x_r = (1+\alpha)x_c - \alpha x_h \qquad f_r = f(x_r).$$

5. (Ветвление) Этот шаг зависит от значения f_r в сравнении с $f_l < f_g < f_h$.

1. Если $f_r < f_l$, то направление вероятно было выбрано удачное. Можно сделать попытку увеличить шаг. Производится растяжение, находится новая точка и значение функции в ней:

   $$x_e = (1-\gamma)x_c + \gamma x_r, \qquad f_e = f(x_e).$$

   Если $f_e < f_r$, то точке x_h присваивается значение x_e, а иначе — значение x_r. После этого итерация заканчивается.
2. Если $f_l < f_r < f_g$, то выбор точки неплохой: точке x_h присваивается значение x_r. После этого итерация заканчивается.
3. Если $f_g < f_r < f_h$, то точки x_r и x_h меняются местами (значения f_r и f_h тоже). После перейти на следующий шаг.
4. Если $f_h < f_r$, просто перейти на следующий шаг.

5. («Сжатие») Строится точка x_s и вычисляется значение функции в ней:

$$x_s = \beta x_h + (1-\beta)x_c, \qquad f_s = f(x_s).$$

6. (Ветвление) Если $f_s < f_h$, то точке x_h присваивается значение x_s. После этого итерация заканчивается.

7. («Глобальное сжатие») Иначе ($f_s > f_h$) имеет место ситуация, когда первоначальные точки оказались самыми удачными. Делается преобразование (сжатия к точке x_i):

$$x_i \leftarrow x_l + (x_i - x_l)/2, i \neq l.$$

8. («Проверка сходимости») Последний шаг — проверка сходимости. Может выполняться по-разному. Если нужная точность не достигнута перейти к пункту 2.

Сингулярное разложение матриц

1. Матричные разложения

Спектральное разложение

Представление матрицы в виде произведения некоторых других, обладающих определенными свойствми, называется матричным разложением. Примером матричного разложения может служить спектральное разложение симметричной матрицы X. По теорема о приведении самосопряженных опреаторов к диагональному виду:

$$X = S^T \cdot D \cdot S,$$

где S — ортогональная матрица, а $D = \mathrm{diag}(\lambda_1, ..., \lambda_i)$ — диагональная матрица из собственных значений матрицы X.

Часто в приложениях встречаются так называемые квадратичные формы, то есть функции вида $f(y) = y^T X y$, где X — симметричная матрица. С помощью спектрального разложения матрицы X можно привести квадратичную форму к более простому виду:

$$f(y) = y^T \cdot S^T \cdot D \cdot S \cdot y = (S \cdot y)^T \cdot D \cdot (S \cdot y) = z^T \cdot D \cdot z = \sum_{i=1}^{n} \lambda_i z_i^2,$$

где была введена естественная замена $z = S \cdot y$.

Сингулярное разложение

В случае произвольной матрицы X имеет место так называемое сингулярное разложение. Пусть X — произвольная матрица, тогда существуют такие ортогональные матрицы U и V, а также диагональная матрица D, что:

$$X = U \cdot D \cdot V.$$

Сингулярное разложение раскрывает геометрическую структуру линейного преобразования, задаваемого матрицей X: представляет его в виде последовательных вращения, рястяжения по осям и еще одного вращения.

Сингулярное разложение имеет множество практических приложений. В том числе и в области анализа данных.

2. Приближение матрицей меньшего ранга

Оценка ранга произведения матриц

Рангом (строчным и столбцовым) матрицы называется соответственно максимальное количество линейно независимых строк или столбцов. Одним из ключевых результатов линейной алгебры является то, что строчный ранг совпадает со столбцовым рангом и равен максимальному размеру невырожденной подматрицы. То есть ранг матрицы $X \in \mathbb{R}^{n \times m}$ размера $n \times m$ не может превосходить ни число строк, ни число столбцов в этой матрице:

$$X \in \mathbb{R}^{n \times m} \implies \mathrm{rg}\,(X) \leq \min(n, m).$$

Пусть теперь матрица $X \in \mathbb{R}^{n \times m}$ представляет собой произведение матриц $A \in \mathbb{R}^{n \times k}$ и $B \in \mathbb{R}^{k \times m}$, причем причем $k < \min(n, m)$. В таком случае $\mathrm{rg}\,(A) \leq k$, $\mathrm{rg}\,(B) \leq k$, а следовательно и для ранга матрицы $X = AB$ будет верна оценка $\mathrm{rg}\,(X) \leq k$.

Ранг матрицы и сжатие без потерь

Более того, всякую матрицу $X \in \mathbb{R}^{n \times m}$ ранга k можно представить в виде произведения матриц A и B размеров $n \times k$ и $k \times m$ соответственно.

Доказательство: Поскольку $\mathrm{rg}\, X = k$, в матрице X существует система из k линейно независимых столбцов и всякий столбец матрицы X представим в виде линейной комбинации столбцов этой системы.

Пусть теперь матрица B составлена из коэффициентов разложения произвольного столбца матрицы X, а матрица A — из вышеупомянутой системы столбцов. Тогда по построению:

$$X = AB, \qquad A \in \mathbb{R}^{n \times k}, \qquad B \in \mathbb{R}^{k \times m}.$$

Это и есть искомое матричное разложение. И поскольку по матрицам A и B исходная матрица X восстанавливается точно, говорят, что происходит сжатие без потерь. Именно поэтому о ранге можно говорить как о мере информационной наполненности матрицы.

Аппроксимация матрицей меньшего ранга

В практических задачах данные всегда измеряются вместе с шумом и поэтому не вся информация, содержащаяся в матрице с данными представляет ценность для исследователя. Таким образом, ставится задача о нахождении лучшей аппроксимации исходной матрицы X некоторой матрицой, ранг которой не превосходит $r < k$. Любая матрица ранга r может быть представлена как произведение следующих матриц:

$$U \cdot V^T, \qquad U \in \mathbb{R}^{m \times k}, \quad V \in \mathbb{R}^{n \times k}.$$

Поэтому удобно переформулировать задачу: требуется найти такие матрицы $U \in \mathbb{R}^{m \times k}$ и $V \in \mathbb{R}^{n \times k}$, что матрица X и UV^T будут отличаться не сильно. Для оценки степени близости объектов используется понятие нормы. Для оценки близости матриц обычно используется так называется норма Фробениуса:

$$\|A\|_2 = \sqrt{\sum_{i,j} |a_{ij}|^2}.$$

В конечном итоге, задача приближения матрицы матрицей меньшего ранга примет вид:

$$U, V = \underset{U \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{n \times k}}{\operatorname{argmin}} \sum_{i,j} \left(x_{ij} - u_i v_j^T\right)^2,$$

а искомой матрицей, дающей наилучшее приближение при заданном ранге, будет матрица UV^T.

Преобразование признаков

Пусть X — матрица признаков объектов. Пусть для X построено наилучшее приближение матрицей UV^T ранга $k < n$, где n — количество признаков в исходной задаче.

Матрица U может быть проинтерпретирована как матрица новых признаков тех же объектов. При этом размерность пространства признаков уменьшается — происходит сжатие с потерями, причем теряется минимум полезной информации.

Задача рекомендации

Пусть X — матрица с оценками, которые поставил или поставил бы пользователь под номером i фильму под номером j. Поскольку далеко не все пользователи смотрели все фильмы и выставили оценку, известны не все элементы этой матрицы.

Чтобы спрогнозировать неизвестные данные, можно попытаться приблизить исходную матрицу с помощью матрицы меньшего ранга. В таком случае в качестве нормы следует использовать норму Фробениуса, где суммирование идет только по известным элементам матрицы X. После того, как наилучшее приближение по известным данным было найдено, эту матрицу можно использовать, чтобы спрогнозировать еще не известные данные.

3. Сингулярное разложение и низкоранговое приближение

Пусть задана матрица X. Требуется найти такую матрицу, ранг которой $\operatorname{rg} \hat{X} \leq k$, которая наилучшим образом приближает исходную:

$$\hat{X} = \underset{\operatorname{rg} \hat{X} \leq k}{\operatorname{argmin}} \left\| X - \hat{X} \right\|$$

SVD для исходной матрицы имеет вид:

$$X = U \cdot D \cdot V^T,$$

где U и V — ортогональные (то есть и невырожденные) матрицы, а D — диагональная матрица ранга $\operatorname{rg} X$. Можно сделать естественную замену искомой матрицы $\hat{X}$ на матрицу $\hat{D}$ (не обязательно диагональную, поэтому это тождественное преобразование):

$$\hat{X} = U \cdot \hat{D} \cdot V^T.$$

Задача перепишется в виде:

$$\hat{X} = \underset{\operatorname{rg} \hat{D} \leq k}{\operatorname{argmin}} \left\| U \cdot (D - \hat{D}) \cdot V^T \right\| = \underset{\operatorname{rg} \hat{D} \leq k}{\operatorname{argmin}} \left\| D - \hat{D} \right\|.$$

Поскольку матрица D — диагональная, все недиагональные элементы в матрице $\hat{D}$ должны быть равными нулю (в ином случае это может только увеличить норму разницы). В матрице $\hat{D}$ может быть максимум k ненулевых элементов (по условию на ранг $\hat{X}$). Поэтому чтобы обеспечить минимум $\left\| D - \hat{D} \right\|$ выберем $\hat{D}$ равной матрице D, в которой все кроме k наибольших по модулю диагональных элементов заменены нулями.

Таким образом, наилучшим приближением матрицы X матрицей ранга k будет

$$\hat{X} = U \cdot \hat{D} \cdot V^T,$$

где матрица $\hat{D}$ — это матрица D, в которой все кроме k наибольших по модулю диагональных элементов заменены нулями.

Следует отметить, что и матричное разложение $X = AB$ определено неоднозначно. В первом случае для любой невырожденной матрицы R нужного размера можно записать:

$$X = AB = AIB = AR^{-1}RB = A'B'.$$

A' и B' тоже будут образовывать некоторое другое разложение матрицы X. Это создает целый ряд проблем при решении задач рекомендаций и подробно будет обсуждаться в соответствующем курсе.

Вероятность и случайные величины

1. Случайность в теории вероятностей и статистике

Большинство явлений окружающего мира слишком сложны для того, чтобы их можно было описать простыми детерминированными законами. Например, предсказания атмосферных явлений не являются абсолютно точными, хотя законы, по которым существуют отдельные молекулы атмосферы, исследованы достаточно хорошо. Тем не менее, во взаимодействии молекул принимают участие слишком много различных факторов, что делает невозможным построение исчерпывающей прогнозирующей модели.

Более простым процессом является подбрасывание кубика, но и здесь точное предсказание результата не представляется возможным. Можно отойти от построения сложной физической модели подбрасывания и перейти к рассмотрению некоего «черного ящика». Этот ящик по неизвестным законам генерирует случайные события, соответствующие числам, выпадающим на кубике. В математике такие «черные ящики» называются *случайными величинами*, а генерируемые события — *реализациями случайной величины*. Набор реализаций случайной величины называется *выборкой* из нее.

Нельзя предугадать, какое событие произойдет в следующий момент наблюдения за «черным ящиком», однако, если вести наблюдения достаточно долго, начнут прослеживаться определенные закономерности. Например, каждое число на кубике будет выпадать примерно одинаковое количество раз. Именно такие закономерности являются объектом изучения теории вероятностей и математической статистики.

Если проводить эксперимент со случайной величиной бесконечно, то каждому событию можно будет поставить в соответствие его *вероятность* — долю испытаний, завершившихся наступлением события (определение нестрогое). Вероятность не может быть измерена на практике в силу данного определения.

Теория вероятностей изучает модели случайных величин и свойства этих моделей. **Статистика и анализ данных** пытаются по свойствам конечных выборок определить свойства случайной величины, чтобы понять, как она будет вести себя в будущем. Осуществить такой переход позволяет *закон больших чисел*: на большой выборке частота события хорошо приближает его вероятность (формулировка нестрогая).

2. Свойства вероятности

Основные свойства вероятности:

1) $0 \leqslant P(A) \leqslant 1$, то есть вероятность любого события лежит на отрезке от нуля до единицы.
2) $P(\varnothing) = 0$ — событие, вероятность которого равна нулю, называется *невозможным*.
3) $P\big(\bar{A}\big) + P(A) = 1$. Для события A всегда можно определить событие «не A», которое соответствует событию «A не произошло». Вероятности таких событий в сумме дают единицу.

Для пары событий возможны следующие отношения:

1) *Вложенность*: $A \subseteq B$. Примером такого отношения является стрельба из арбалета по мишени. Событие A — это попадание в «десятку», событие B — набор пяти или более очков. Тогда событие A вложено в событие B, причем мишень визуализирует это буквально (см. рис. 1).

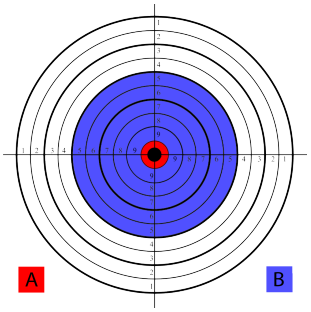

Рис. 1.

Вероятности таких событий связаны друг с другом неравенством:

$$A \subseteq B \quad \Rightarrow \quad P(A) \leqslant P(B).$$

2) *Произведение событий* AB и *сумма событий* $A + B$. Пусть событие A — это попадание в синюю область на мишени, событие B — в зеленую (см. рис. 2). Тогда событие AB — это пересечение этих областей, то есть произошло каждое из данных событий (см. рис. 3).

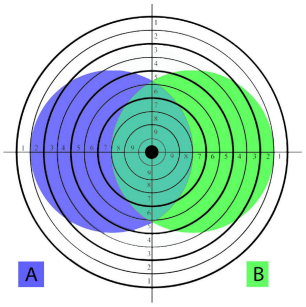

Рис. 2.

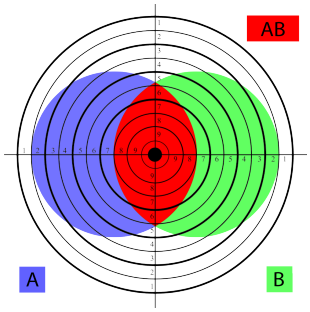

Рис. 3.

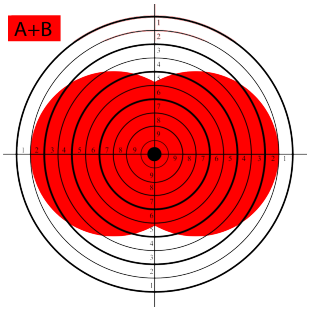

Рис. 4.

Сумма событий $A+B$ означает попадание в область, соответствующую объединению этих двух областей, то есть произошло хотя бы одно из данных событий (см. рис. 4).

Вероятности событий AB и $A + B$ связаны следующим образом:

$$P(A + B) = P(A) + P(B) - P(AB).$$

3) *Дополнение*: $B\backslash A$. Здесь происходит событие B, но не происходит событие A. Вероятность такого события задаётся следующим выражением:

$$P\big(B\backslash A\big) = P(B) - P(AB).$$

В случае, когда A полностью лежит в B, формула упрощается:

$$P\big(B\backslash A\big) = P(B) - P(A).$$

В примере из пункта 1 это соответствует попаданию в область от пяти до девяти очков (см. рис. 5).

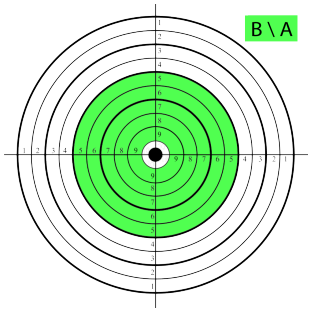

Рис. 5.

4) *Независимость событий*: $P(AB)$. Пусть событие A — это попадание в нижнюю половину мишени, а событие B — попадание в правую половину. Событием AB будет пересечение этих двух событий — попадание в нижнюю правую четверть (см. рис. 6).

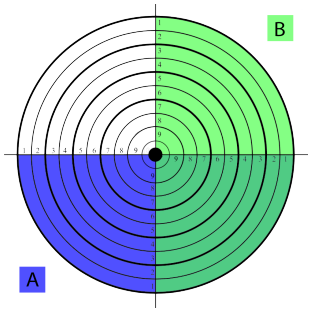

Рис. 6.

События A и B являются *независимыми*, если вероятность их пересечения равна произведению вероятностей компонент:

$$P(AB) = P(A)P(B).$$

В примере с мишенью, если можно считать, что арбалет хорошо настроен и не дует ветер, то:

$$P(A) = 0{,}5, \quad P(B) = 0{,}5, \quad P(AB) = 0{,}25,$$

что означает независимость двух рассматриваемых событий.

3. Условная вероятность

Пусть событие A в примере с мишенью — это попадание в «десятку», событие B — попадание в любое место мишени. Если известно, что событие B произошло, то вероятность события A повышается. *Условная вероятность* события A при условии, что произошло событие B, определяется следующим образом:

$$P\big(A|B\big) = \frac{P(AB)}{P(B)}.$$

Если попадание в мишень происходит в восьмидесяти процентах случаев, а в «десятку» — в пяти процентах, то:

$$P(AB) = P(A) = 0{,}05 \quad \Rightarrow \quad P\big(A|B\big) = \frac{0{,}05}{0{,}8} = 0{,}0625.$$

Формула полной вероятности имеет вид:

$$P(A) = P\big(A|B\big)P(B) + P\big(A|\bar{B}\big)P\big(\bar{B}\big).$$

Условные вероятности двух событий связаны друг с другом с помощью **формулы Байеса**:

$$P\big(A|B\big) = \frac{P(A)P(B|A)}{P(B)}.$$

4. Дискретные случайные величины

В испытании с подбрасыванием кубика возможны шесть исходов. Эти исходы можно пронумеровать $1, 2, 3, 4, 5, 6$ согласно значению, выпавшему на кубике. Данному множеству исходов ставится в соответствие вектор вероятности:

$$\{1, 2, 3, 4, 5, 6\} \quad \Longrightarrow \quad \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{pmatrix},$$

причем:

$$p_i \geqslant 0, \;\; i = 1, \ldots, 6; \quad \sum_{i=1}^{6} p_i = 1.$$

Именно так устроена *дискретная случайная величина* X. Она принимает счетное множество значений $A = \{a_1, a_2, a_3, \ldots\}$ с вероятностями $p_1, p_2, p_3, \ldots$, где:

$$p_i \geqslant 0 \;\forall i, \quad \sum_{i=1}^{\infty} p_i = 1.$$

Вероятность того, что $X = a_i$, равна p_i, тогда *функция вероятности* имеет вид:

$$P\big(X = a_i\big) = p_i.$$

Одной из наиболее часто встречающихся случайных величин является **дискретная случайная величина с двумя исходами**. Примером в данном случае является подбрасывание монеты. Можно обозначить выпадение решки за 1 («успех»), а выпадение орла — за 0 («неудача»). Пусть вероятность «успеха» равна p:

$$P(X = 1) = p, \quad P(X = 0) = 1 - p.$$

Именно так устроена *бернуллиевская случайная величина*:

$$X \sim \text{Ber}(p).$$

Другим примером является **сумма независимых бинарных случайных величин**. Пусть вероятность попадания мяча в баскетбольное кольцо равна p, имеется n попыток, а число попаданий равно X. В силу независимости попыток:

$$P(X = n) = p^n.$$

В общем случае:

$$P(X = k) = C_n^k p^k (1 - p)^{n-k}, \ k = 0, 1, 2, \ldots, n.$$

Здесь используется *биномиальный коэффициент*:

$$C_n^k = \frac{n!}{k!(n-k)!}.$$

Такая случайная величина X называется *биномиальной*. Биномиальное распределение имеет два параметра: целочисленный n и $p \in [0, 1]$:

$$X \sim \text{Bin}(n, p).$$

Еще одним классом дискретных случайных величин являются **счетчики**. В качестве иллюстрации можно рассмотреть текст романа Набокова «*Приглашение на казнь*». Из всех произведений Набокова составляется словарь, затем текст данного романа сверяют с этим словарем. Некоторые слова не встречаются в романе ни разу, например, слово «шахматист». Некоторые слова встречаются в романе редко, например, слово «аляповатость» (1 раз). Есть и слова, которые используются очень часто (например, «год» — 21 раз).

Пусть X — это число использований слова в тексте. Вероятность того, что X равно k, можно описать *распределением Пуассона*:

$$P(X = k) = \frac{\lambda^k \, \mathrm{e}^{-\lambda}}{k!}, \quad \lambda > 0, \quad k = 0, 1, 2, \ldots.$$

$$X \sim \text{Pois}(\lambda).$$

Распределением Пуассона описывается, например, число автобусов, которые проезжают за час мимо автобусной остановки, или число радиоактивных распадов, которые улавливает счетчик Гейгера.

5. Непрерывные случайные величины

Непрерывные случайные величины нельзя задавать с помощью функции вероятности в силу того, что если множество A несчетное, то вероятность события нулевая:

$$|A| > \aleph_0 \quad \Rightarrow \quad P(X = a) = 0 \ \forall a \in A.$$

Первый способ определения таких величин — с помощью *функции распределения* (см. рис. 7):

$$F(x) = P(X \leqslant x).$$

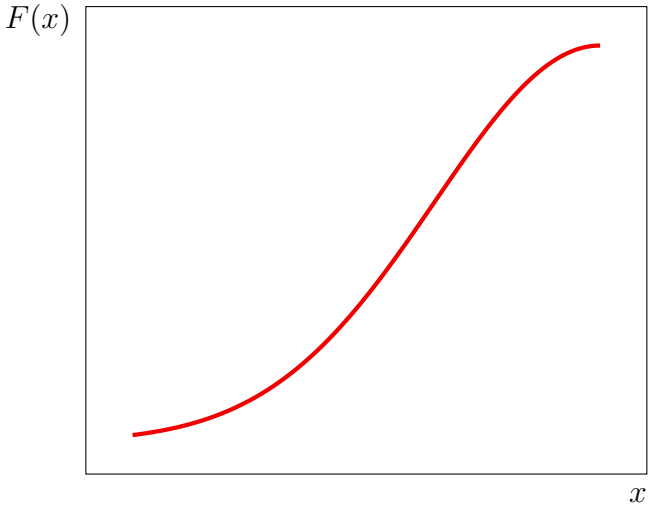

Рис. 7.

Функция распределения всегда принимает значение от 0 до 1 и не убывает по аргументу x.

Второй способ определения непрерывных случайных величин — с помощью *плотности распределения*:

$$f(x) : \int_a^b f(x)\,dx = P(a \leqslant X \leqslant b).$$

Плотность связана с функцией распределения следующим образом:

$$F(x) = \int_{-\infty}^{x} f(u)\,du.$$

Для непрерывной случайной величины верно равенство:

$$\int_{-\infty}^{+\infty} f(u)\,du = P(-\infty \leqslant X \leqslant +\infty) = 1.$$

В отличие от функции распределения, плотность распределения на графике может принимать различный вид.

Примером непрерывной случайной величины является *равномерная случайная величина*. Пусть X — это время ожидания на светофоре до того, как можно будет перейти дорогу. Если на светофоре нет счетчика, то нельзя угадать, сколько именно придется ждать зеленого сигнала. Время ожидания может быть любым числом от 0 до, например, 30 секунд. Именно так устроено *равномерное распределение* — случайная величина на отрезке $[a, b]$ принимает любое значение с одинаковой вероятностью:

$$X \sim U(a, b).$$

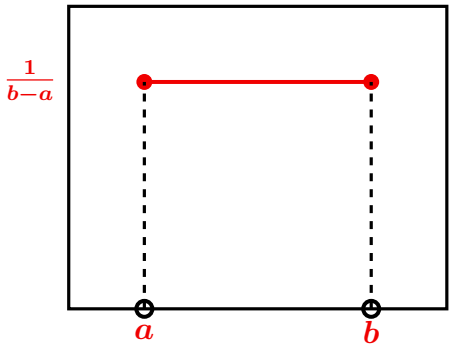

Рис. 8.

Плотность вероятности для равномерной случайной величины имеет вид (см. рис. 8):

$$f(x) = \begin{cases} \dfrac{1}{b-a}, & x \in [a, b], \\ 0, & x \notin [a, b]. \end{cases}$$

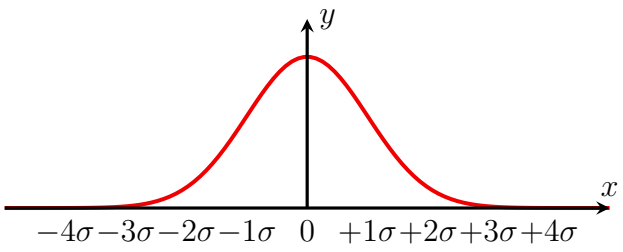

Рис. 9.

Другим примером непрерывной случайной величины является *нормальная случайная величина*. Человек из примера любит приходить на работу к 11 часам, но точное время его прихода варьируется — иногда он проснется раньше, иногда он опаздывает. Таким образом, точное время его прихода на работу X представляет собой результат взаимодействия большого количества слабо зависимых случайных факторов. Именно такие величины хорошо моделируются *нормальным (Гауссовым) распределением*:

$$X \sim N\left(\mu, \sigma^2\right).$$

В данном примере параметр μ отвечает за среднее время прихода, а параметр σ определяет разброс вокруг среднего. Функция плотности вероятности нормального распределения имеет вид (см. рис. 9):

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

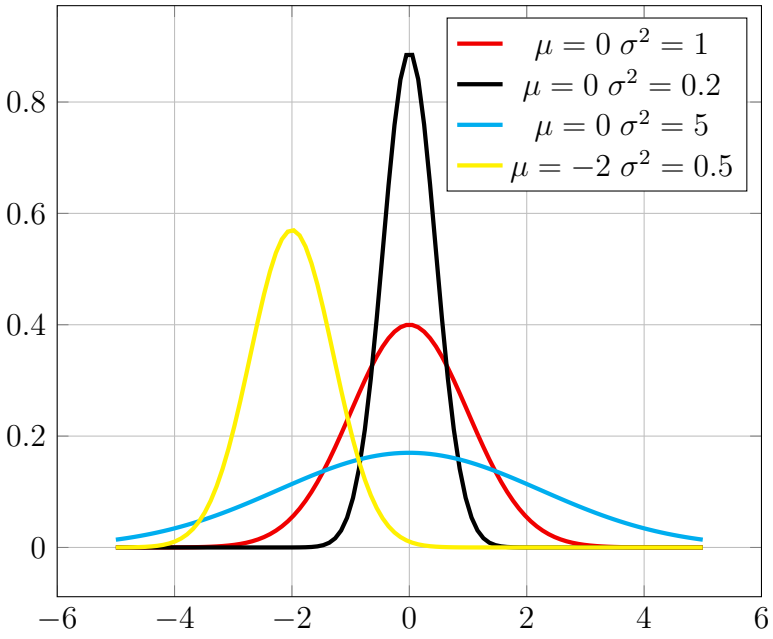

Рис. 10.

Варьируя значения параметров μ и σ, можно влиять на форму графика функции плотности вероятности нормального распределения (см. рис. 10).

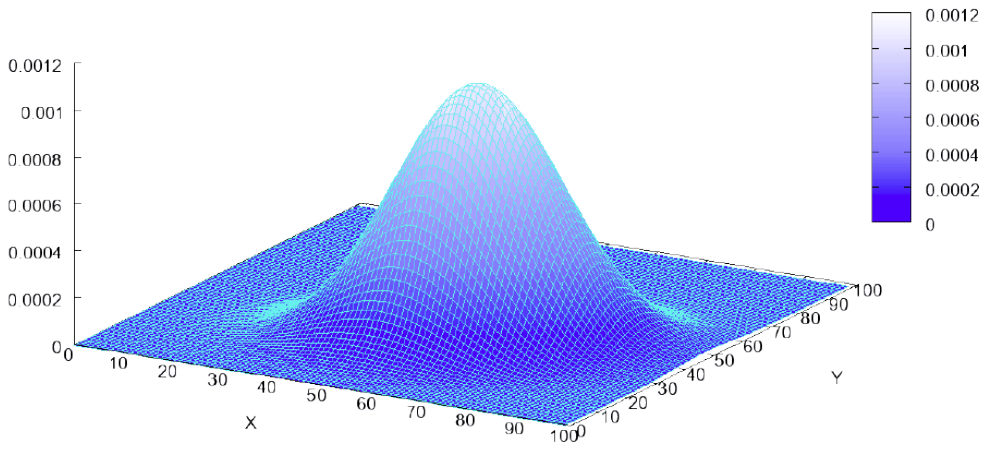

Рис. 11.

Нормальное распределение может быть *многомерным*. В этом случае случайная величина является не скалярной, а векторной:

$$X \sim N(\mu, \Sigma), \quad \mu \in \mathbb{R}^k, \Sigma \in \mathbb{R}^{k\times k},$$

причем матрица Σ является положительно определенной. Функция плотности вероятности в данном случае имеет вид (см. рис. 11):

$$f(x) = (2\pi)^{-\frac{k}{2}} \left|\Sigma\right|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(x-\mu)^T|\Sigma|^{-1}(x-\mu)\right).$$

Статистики

1. Оценка распределения по выборке

Рассматривается выборка из случайной величины X:

$$X^n = (X_1, \dots, X_n),$$

где n — объем выборки. Величины $X_1, X_2, \dots, X_n$ — независимые одинаково распределенные случайные величины (*i.i.d.*).

Статистикой $T(X^n)$ называется любая функция от данной выборки.

Далее будет рассмотрено, какие статистики используются для оценок по выборкам законов распределения случайных величин различных классов. Распределение *дискретной случайной величины* задается функцией вероятности:

$$X \in A = \{a_1, a_2, \dots\}, \quad P(X = a_k) = p_k.$$

Для выборки из такой случайной величины лучшей оценкой для вероятностей из функции вероятностей являются частоты соответствующих событий на выборке (по закону больших чисел):

$$\bar{p}_k = \frac{1}{n} \sum_{i=1}^{n} [X_i = a_k].$$

Если *непрерывная случайная величина* задается с помощью функции распределения, то ее можно оценить с помощью **эмпирической функции распределения**:

$$X \sim F(x), \quad F_n(x) = \frac{1}{n} \sum_{i=1}^{n} [X_i \leqslant x].$$

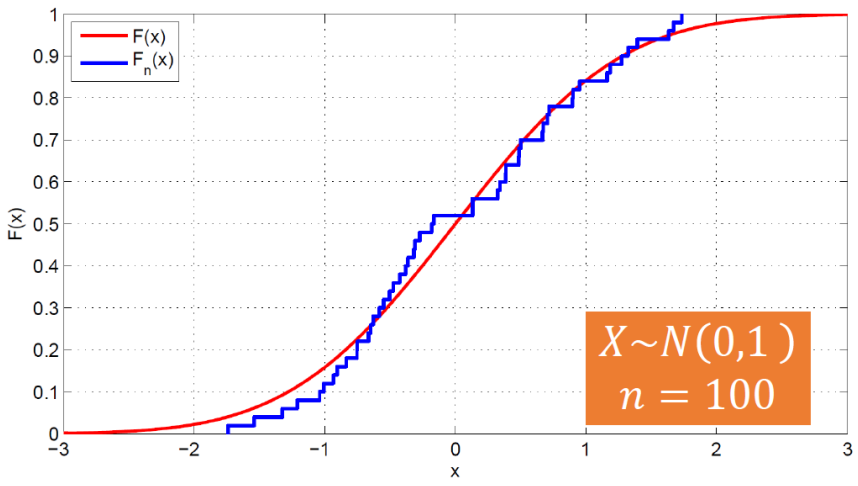

Рис. 1.

На рис. 1 красная линия соответствует теоретической функции стандартного нормального распределения (нормальное распределение со средним, равным нулю, и с дисперсией, равной 1). Синяя линия соответствует эмпирической функции распределения, построенной по выборке объема 100.

Непрерывные случайные величины также могут задаваться с помощью плотностей. Для оценки плотности можно разбить область определения случайной величины на интервалы одинаковой длины. Количество объектов выборки в каждом интервале будет пропорционально среднему значению плотности на нем. Именно так устроена **гистограмма**.

На гистограмме, приведенной на рис. 2, изображена продолжительность жизни крыс на строгой диете (в днях).

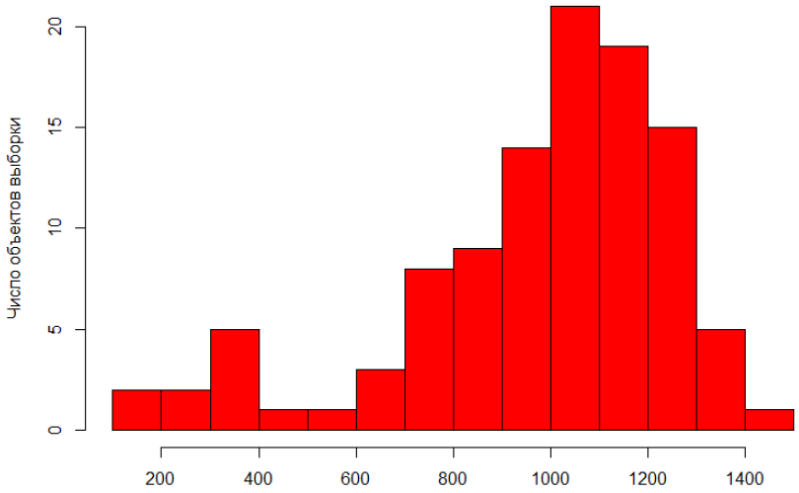

Рис. 2.

По такой гистограмме хорошо видны все особенности распределения данных: оно бимодально, основной пик приходится примерно на 1000 дней, но есть крысы, которые живут существенно меньше.

Важным аспектом работы с гистограммами является правильный выбор числа интервалов.

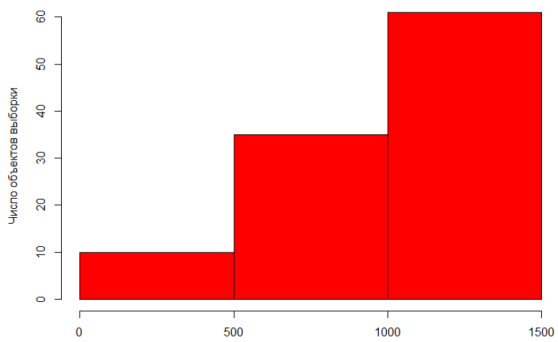

Рис. 3.

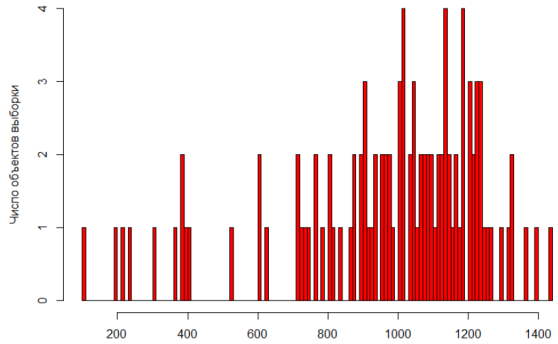

Рис. 4.

Если рассмотреть слишком мало интервалов, то они будут слишком большими, в результате гистограмма получится грубой (см. рис. 3). Аналогично в случае слишком большого количества интервалов — в большую часть из них не попадет ни одного объекта выборки (см. рис. 4). В обоих случаях построенные гистограммы не являются информативными.

Описанного недостатка лишены *гладкие оценки плотности* $f(x)$. Для построения такой оценки необходимо взять окно ширины h и, двигая его по числовой оси, вычис-

лять в нем значение функции, называемой **ядром**. **Ядерная оценка плотности** имеет вид:

$$f_n(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{X_i - x}{h}\right).$$

На рис. 5 показана оценка, построенная на тех же данных о продолжительности жизни крыс. Как и в случае гистограммы, на таком графике видны все особенности распределения данных.

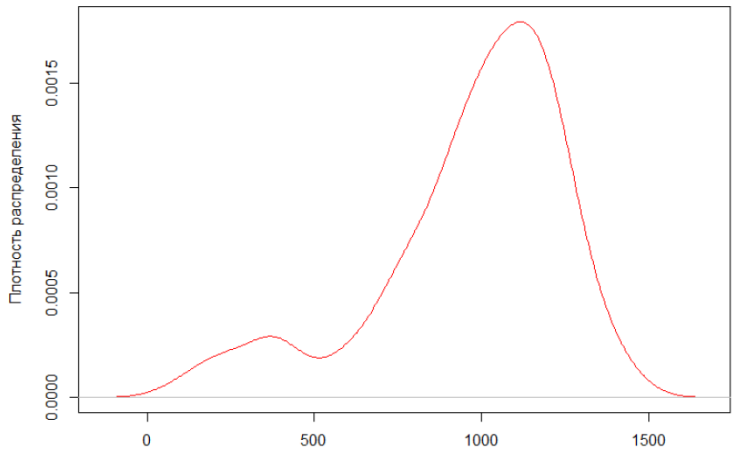

Рис. 5.

На рис. 6 продемонстрированы все виды оценок распределения для выборки из стандартного нормального распределения.

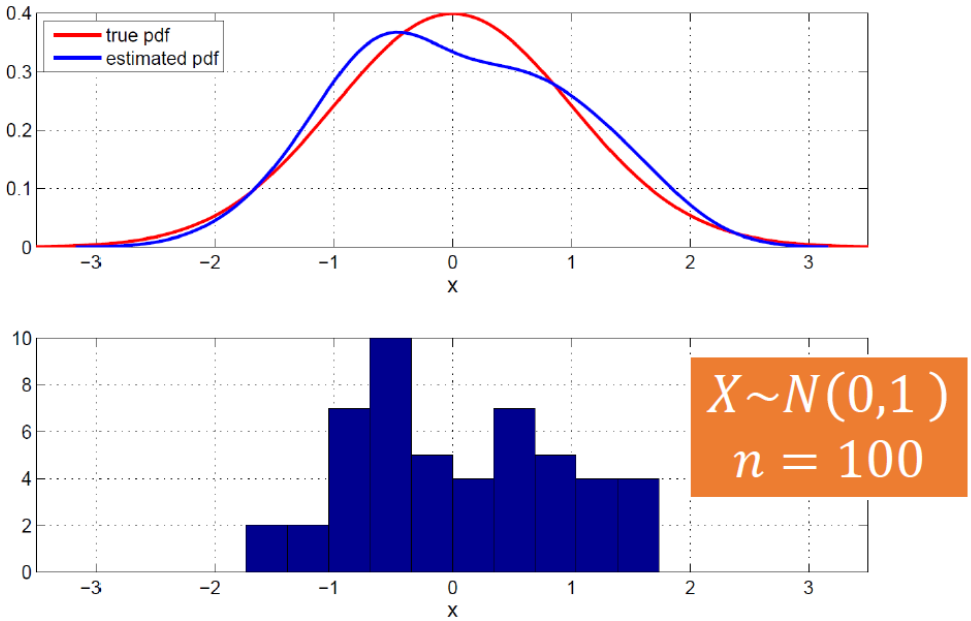

Рис. 6.

Необходимо отметить, что ни один из представленных способов оценки плотностей не является идеальным, так что рекомендуется использовать оба.

2. Важные характеристики распределений

Часто возникает необходимость оценить не всю функцию распределения, а некоторые ее параметры. Самым важным классом параметров распределения являются **средние**. Нестрогое определение можно сформулировать следующим образом: среднее — это значение, вокруг которого группируются все остальные.

Одним из вариантов уточнения данного определения является **матожидание**:

$$EX = \begin{cases} \sum\limits_i a_i p_i, & X \text{ — дискретна,} \\ \int\limits_{-\infty}^{+\infty} x\, f(x)\, dx, & X \text{ — непрерывна.} \end{cases}$$

Другой характеристикой среднего является медиана. Она определяется с помощью квантиля. **Квантилем** порядка $\alpha \in (0,1)$ называется величина X_α такая, что:

$$P\big(X \leqslant X_\alpha\big) \geqslant \alpha, \quad P\big(X \geqslant X_\alpha\big) \geqslant 1-\alpha.$$

Медиана — это квантиль порядка 0,5:

$$P\big(X \leqslant \operatorname{med} X\big) \geqslant 0{,}5, \quad P\big(X \geqslant \operatorname{med} X\big) \geqslant 0{,}5.$$

Еще одной характеристикой среднего является **мода** — самое вероятное значение случайной величины (в нестрогом смысле):

$$\operatorname{mode} X = \begin{cases} a_{\operatorname{argmax}\limits_i p_i}, & X \text{ — дискретна,} \\ \operatorname{argmax}\limits_x f(x), & X \text{ — непрерывна.} \end{cases}$$

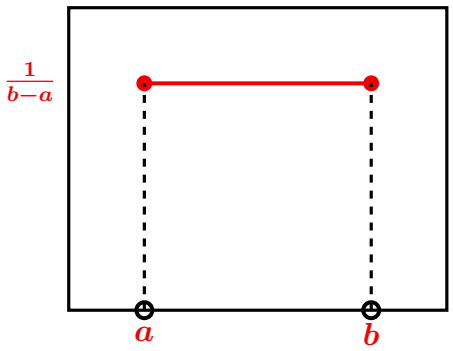

Рис. 7.

В случае нормально распределенной случайной величины ее матожидание, медиана и мода в точности совпадают:

$$X \sim N\big(\mu, \sigma^2\big) \quad \Rightarrow \quad EX = \operatorname{med} X = \operatorname{mode} X = \mu.$$

Если случайная величина X равномерно распределена на отрезке [a,b], то ее матожидание и медиана совпадают:

$$X \sim U(a,b) \quad \Rightarrow \quad EX = \operatorname{med} X = \frac{a+b}{2}.$$

Мода такой случайной величины не определена, поскольку у плотности распределения нет максимума (см. рис. 7). Значит, модой в данном случае может быть любое число на интервале от a до b.

В случае бимодального распределения мода приходится на максимум плотности, а медиана и матожидание смещены в сторону второго «горба», причем смещение матожидания больше, чем смещение медианы (см. рис. 8).

Рис. 8.

Следующая рассматриваемая группа параметров распределения — это параметры, характеризующие **разброс**, то есть то, насколько случайная величина концентрируется вокруг своего среднего значения. Одним из наиболее важных параметров здесь является **дисперсия**:

$$DX = E\Big((X - EX)^2\Big).$$

Часто используется величина $\sqrt{DX}$, называемая **среднеквадратическое отклонение**.

Еще одна характеристика разброса — **интерквантильный размах**:

$$\text{IQR} = X_{0,75} - X_{0,25}.$$

Если случайная величина X распределена по закону Пуассона с параметром λ, то её дисперсия и матожидание совпадают:

$$X \sim \text{Pois}(\lambda) \quad \Rightarrow \quad DX = \lambda, \quad EX = \lambda.$$

Для нормально распределенной случайной величины дисперсия — это второй параметр распределения:

$$X \sim N(\mu, \sigma^2) \quad \Rightarrow \quad DX = \sigma^2.$$

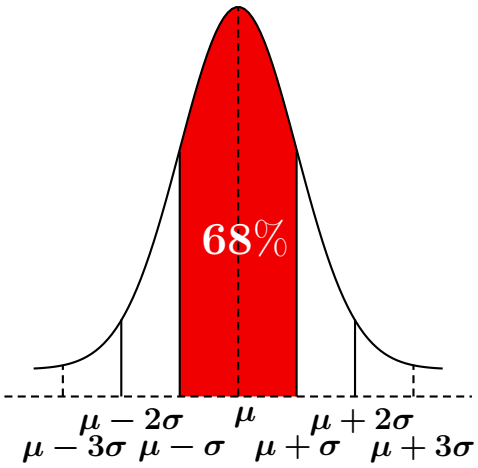

Рис. 9.

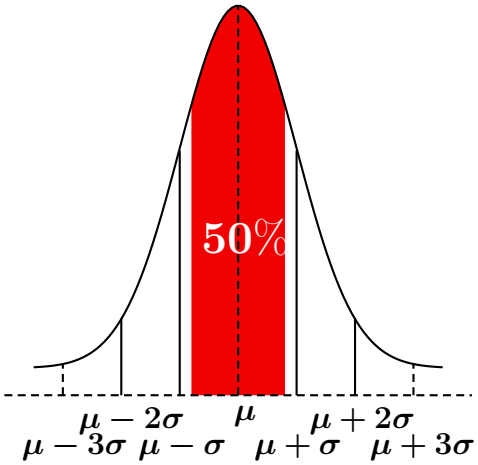

Рис. 10.

В отличие от характеристик среднего значения, характеристики разброса для нормального распределения не совпадают. Для иллюстрации можно отложить от среднего значения μ интервалы, соответствующие среднеквадратическим отклонениям σ. В интервале от $\mu-\sigma$ до $\mu+\sigma$ лежит 68% вероятностной массы нормального распределения (см. рис. 9). В интервал, соответствующий интерквартильному размаху вокруг среднего, попадает 50% случайной величины (см. рис. 10).

В интервал от $\mu-2\sigma$ до $\mu+2\sigma$ попадает примерно 95% вероятностной массы нормально распределенной случайной величины (см. рис. 11). Это часто используемое на практике *правило двух сигм*. Другой его вариант — *правило трех сигм* (см. рис. 12): в интервале от $\mu-3\sigma$ до $\mu+3\sigma$ случайная величина реализуется практически со стопроцентной вероятностью (99,7%).

3. Важные статистики

Оценка матожидания случайной величины — это **выборочное среднее**:

$$\bar{X}=\frac{1}{n}\sum_{i=1}^{n}X_i.$$

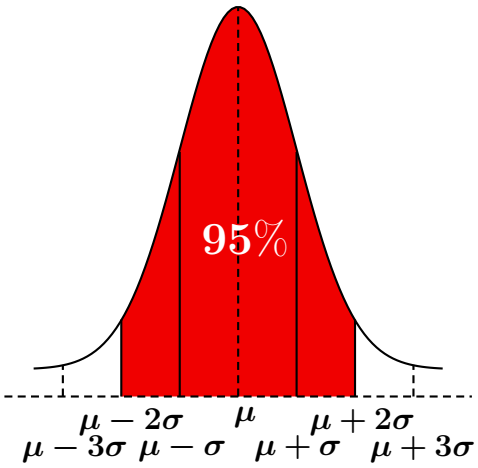

Рис. 11.

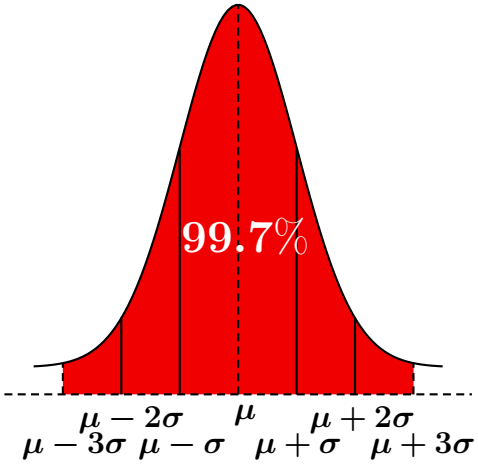

Рис. 12.

Для построения выборочной медианы необходимо составить из рассматриваемой выборки вариационный ряд:

$$X^n = \big(X_1, X_2, \ldots, X_n\big) \quad \Rightarrow \quad X_{(1)} \leqslant X_{(2)} \leqslant \ldots \leqslant X_{(n)}.$$

Элемент вариационного ряда i называется *i-й порядковой статистикой*. **Выборочная медиана** является центральным элементом вариационного ряда:

$$m = \begin{cases} X_{(k)}, & n = 2k+1, \\ \dfrac{X_{(k)} + X_{(k+1)}}{2}, & n = 2k. \end{cases}$$

Выборочная мода оценивается по максимуму оценки плотности распределения.

Показателен следующий пример. Рассматривается выборка из 25 человек, для каждого из которых известен годовой доход. В выборке есть десять человек, годовой доход которых равен двум тысячам долларов, один человек с годовым доходом в три тысячи долларов, и так далее. Один человек получает сорока пять тысяч долларов в год. Среднее арифметическое годовых доходов на этой выборке — 5700 долларов. Здесь медиана составляет 3000 долларов, а мода — 2000.

Необходимо заметить, что все рассматриваемые величины называются «средними». Значит, для оптимистичного отчета по данной выборке можно воспользоваться средним арифметическим, а для пессимистичного — модой.

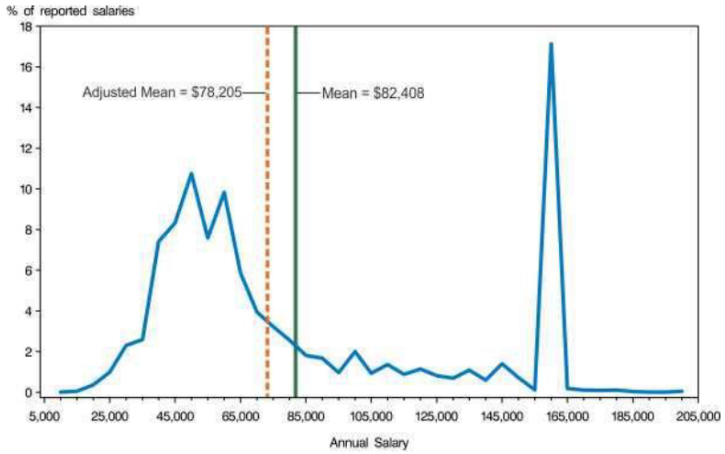

Рис. 13.

Выборочная дисперсия оценивает дисперсию и имеет следующий вид:

$$S^2 = \frac{1}{n-1}\sum_{i=1}^{n}\big(X_i - \bar{X}\big)^2.$$

Для построения выборочной оценки интерквартильного размаха необходимо определить *выборочный квантиль* порядка α: — это порядковая статистика, порядок которой равен целой части от αn ($X_{([\alpha n])}$). Тогда **выборочный интерквартильный размах** определяется следующим образом:

$$\mathrm{IQR}_n = X_{([0{,}75n])} - X_{([0{,}25n])}.$$

В качестве следующего примера будет рассмотрено распределение годового дохода членов американской ассоциации юристов (см. рис. 13). Имеются два выраженных пика — 168 тысяч долларов и 45 тысяч. Среднее значение, посчитанное по данной выборке, равно 82 тысячам долларов. Видно, что это значение несет очень мало информации о выборке, так как крайне мало людей получают именно такой доход.

Другой показательный пример — *квартет Энскомба* (см. рис. 14). Рассматриваются четыре искусственно сгенерированных пары выборок, характеристики которых в каждом из четырех случаев совпадают (равны выборочные средние и выборочные дисперсии). Однако, при рассмотрении диаграмм рассеяния по этим четырем парам выборок, видно, что в каждом из четырех случаев происходят абсолютно разные вещи (см. рис. 15). Таким образом, даже совокупность статистик не позволяет полностью понять данные, и рекомендуется всегда при анализе данных изучать графики, гистограммы и оценки плотности вместо одних лишь цифр.

4. Центральная предельная теорема

Рассмотрим случайную величину X с функцией распределения $F(x)$. Пусть имеется ее выборка объема n:

$$X \sim F(x), \quad X^n = \big(X_1, X_2, \ldots, X_n\big),$$

№	1	2	3	4
$\bar{x}$	9	9	9	9
S_x	11	11	11	11
$\bar{y}$	7.5	7.5	7.5	7.5
S_y	4.13	4.13	4.13	4.13

Рис. 14.

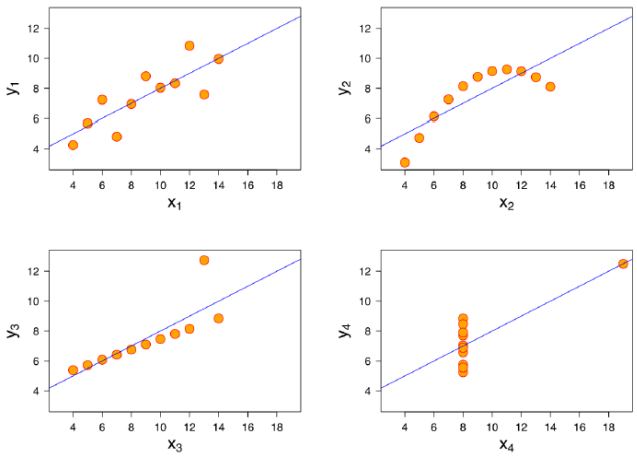

Рис. 15.

По выборке можно вычислить выборочное среднее:

$$\bar{X}_n = \frac{1}{n}\sum_{i=1}^{n} X_i.$$

Какое распределение имеет выборочное среднее, и как оно связано с исходным распределением?

Можно провести эксперимент. Берется случайная величина с распределением, показанным на рис. 16.

Из данной случайной величины можно взять выборку объема n и посчитать по ней выборочное среднее. Данное действие необходимо повторить в рамках эксперимента достаточно много раз, чтобы затем построить гистограмму полученных выборочных средних. На рис. 17 приведена гистограмма, построенные по выборкам объема $n = 2$. По сравнению с исходной плотностью случайной величины, данная гистограмма выглядит более гладкой. С увеличением объёма выборки процесс сглаживания продолжается (см. рис. 18 для $n = 3$).

При объеме выборки $n = 5$ гистограмма становится унимодальной (см. рис. 19). Дальнейшее увеличение выборки не влияет на форму гистограммы, она лишь становится более узкой (см. рис. 20 для $n = 30$).

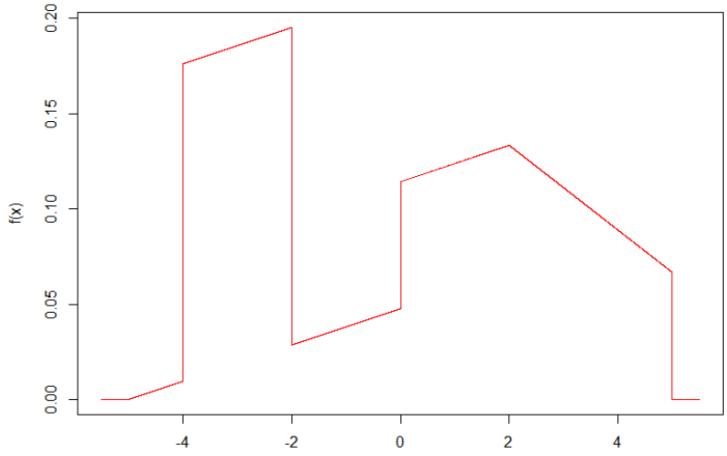

Рис. 16.

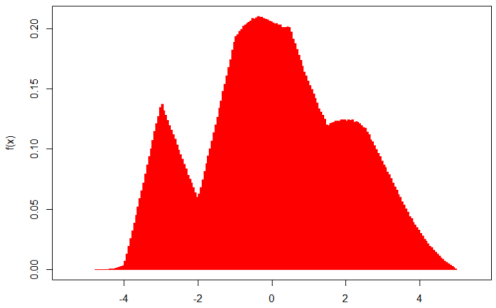

Рис. 17.

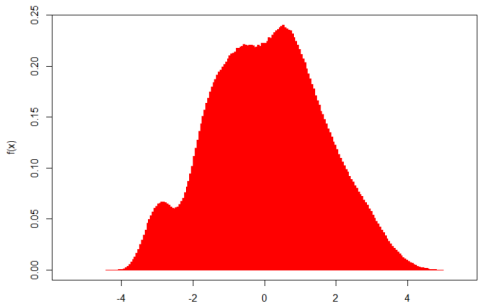

Рис. 18.

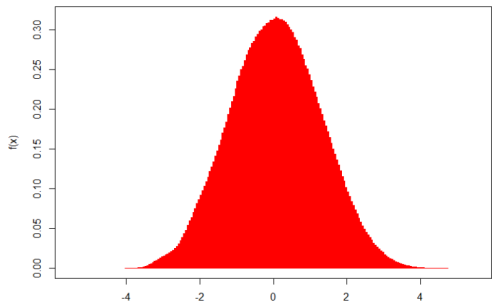

Рис. 19.

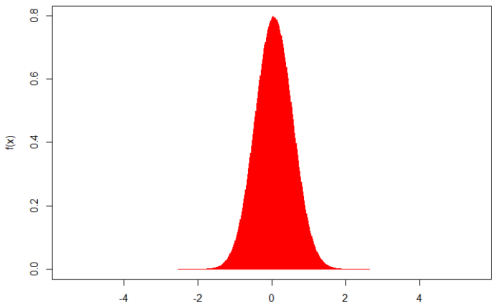

Рис. 20.

Можно заметить, что распределение выборочных средних достаточно хорошо описывается нормальным распределением, что является утверждением **центральной предельной теоремы**:

$$X \sim F(x), \quad X^n = (X_1, X_2, \ldots, X_n) \quad \Rightarrow \quad \bar{X}_n \approx\sim N\left(EX, \frac{DX}{n}\right).$$

С ростом n точность нормальной аппроксимации увеличивается.

Полученный результат справедлив не только для непрерывных распределений, но и для дискретных. Это можно рассмотреть на примере биномиального распределения (см. рис. 21). Как и в предыдущем эксперименте, можно повторить данный несколько раз и построить гистограммы для различного объема выборок (см. рис. 22 для $n = 2$).

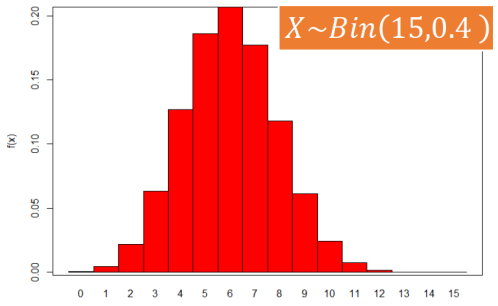

Рис. 21.

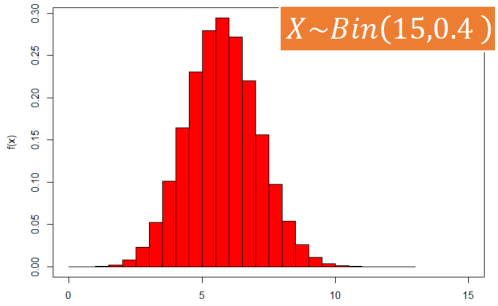

Рис. 22.

При увеличении объема выборок происходит то же, что и в предыдущем эксперименте — распределение становится все более гладким и все более похожим на нормальное (см. рис. 23 для $n = 5$ и рис. 24 для $n = 30$). Таким образом, центральная предельная теорема в данном случае работает.

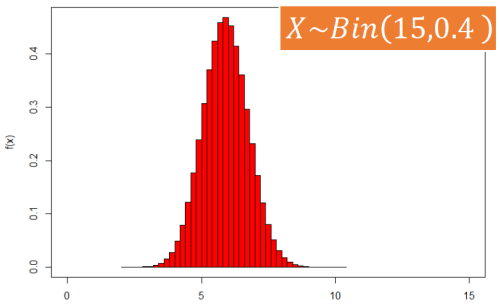

Рис. 23.

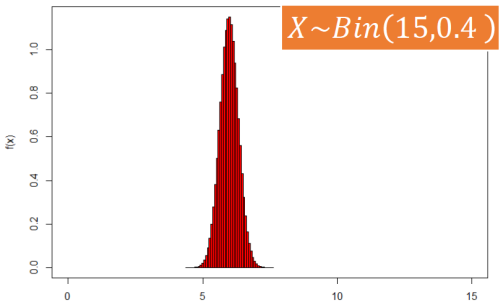

Рис. 24.

Проведём еще один эксперимент для биномиального распределения, на этот раз взяв $p = 0{,}01$. Функция вероятности данной случайной величины показана на рис. 25. На рис. 26 изображено распределение выборочных средних, построенных по выборкам объема $n = 2$.

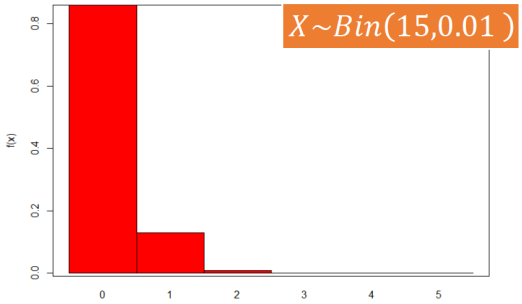

Рис. 25.

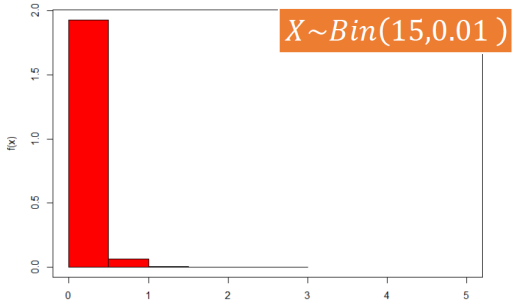

Рис. 26.

Исходное распределение не позволяет распределению выборочных средних быстро сходиться к нормальному (см. рис. 27 для $n = 5$ и рис. 28 для $n = 30$). Даже по выборке объема $n = 30$ гистограмма не может быть хорошо описана нормальным законом — даже относительно максимума данной гистограммы распределение несимметрично.

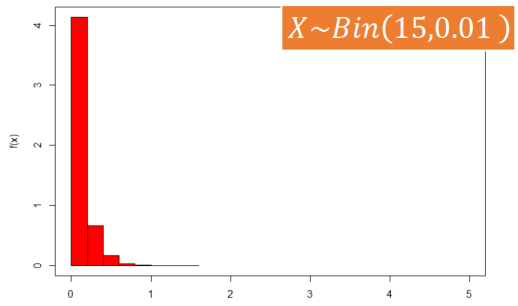

Рис. 27.

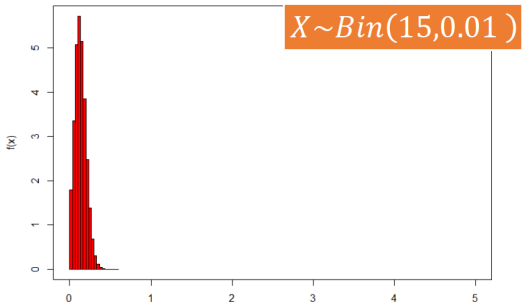

Рис. 28.

Центральная предельная теорема хорошо работает, если исходное распределение не слишком скошено. Существует эмпирическое правило: когда распределение X не слишком скошено, распределение X_n хорошо описывается нормальным при $n \geqslant 30$.

5. Доверительные интервалы

Имеется некий продукт, для которого известна его целевая аудитория. Необходимо узнать, насколько хорошо целевая аудитория знакома с данным продуктом. Введём следующую случайную величину:

$$X = \begin{cases} 1, & \text{член ЦА знает продукт,} \\ 0, & \text{не знает.} \end{cases}$$

Такая случайная величина имеет распределение Бернулли с параметром p — *узнаваемостью продукта*:

$$X \sim \text{Ber}(p).$$

Измерить узнаваемость продукта можно с помощью опроса. Если в опросе n участников, то на выходе получится выборка X^n, состоящая из 0 и 1. Оценкой узнаваемости по данной выборке будет выборочное среднее:

$$\bar{p}_n = \frac{1}{n}\sum_{i=1}^{n} X_i = \bar{X}_n.$$

Пусть по итогам первого опроса, в котором приняло участие 10 человек, оказалось, что 6 из них знакомы с продуктом:

$$n = 10, \quad \bar{p}_n = 0{,}6.$$

Второй опрос дал следующий результат:

$$n = 100, \quad \bar{p}_n = 0{,}44.$$

Необходимо определить, какая из двух полученных оценок лучше. Точность измеренной оценки определяется с помощью **доверительных интервалов**: пары статистик C_L, C_U такой, что:

$$P\big(C_L \leqslant \theta \leqslant C_U\big) \geqslant 1 - \alpha.$$

Здесь θ — это *оцениваемый параметр*, $(1-\alpha)$ — *уровень доверия*, а C_L и C_U — *верхний* и *нижний доверительные пределы* (соответственно). При бесконечном повторении эксперимента в $100(1-\alpha)\%$ случаев этот интервал будет накрывать истинное значение параметра θ.

Доверительные интервалы можно построить для оценок узнаваемости продукта из рассматриваемого примера. Оценки узнаваемости являются, по сути, выборочными средними. Следовательно, можно воспользоваться центральной предельной теоремой:

$$\bar{p}_n \approx\sim N\left(EX, \frac{DX}{n}\right).$$

Для случайной величины с распределением Бернулли известно, что:

$$X \sim \text{Ber}(p) \quad \Rightarrow \quad EX = p, \quad D = p(1-p),$$

тогда:

$$\bar{p}_n \approx\sim N\left(p, \frac{p(1-p)}{n}\right).$$

В правую часть данного выражения можно подставить $\bar{p}_n$ вместо p:

$$\bar{p}_n \approx\sim N\left(\bar{p}_n, \frac{\bar{p}_n(1-\bar{p}_n)}{n}\right).$$

Распределение стало полностью определенным. Далее необходимо воспользоваться правилом двух сигм:

$$\sigma = \sqrt{\frac{\bar{p}_n(1-\bar{p}_n)}{n}} \quad \Rightarrow \quad P\left(\bar{p}_n - 2\sqrt{\frac{\bar{p}_n(1-\bar{p}_n)}{n}} \leqslant p \leqslant \bar{p}_n + 2\sqrt{\frac{\bar{p}_n(1-\bar{p}_n)}{n}}\right) \approx 0{,}95.$$

Применение полученного выражения к двум опросам даст следующий результат. В первом опросе 95% доверительный интервал — $(0{,}29, 0{,}91)$, во втором — $(0{,}34, 0{,}54)$. Таким образом, доверительный интервал помогает в описании степени неуверенности в полученной оценке.

Доверительные интервалы не обязательно строить с помощью центральной предельной теоремы. Для конкретных распределений существуют более точные способы. Например, для распределения Бернулли наиболее точен метод Уилсона. Однако, именно центральная предельная теорема является универсальным средством построения доверительных интервалов — она работает вне зависимости от того, из какого распределения взята исходная выборка.