НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ЯДЕРНЫЙ УНИВЕРСИТЕТ «МИФИ» (НИЯУ МИФИ)
ОБНИНСКИЙ ИНСТИТУТ АТОМНОЙ ЭНЕРГЕТИКИ (ИАТЭ)

Технологии программирования для больших данных

Обзор языка программирования Python

О.А.Мирзеабасов

Отделение ИКС ИАТЭ НИЯУ МИФИ

Осень 2023 – весна 2024

Пользовательские интерфейсы

Вызов интерпретатора Python в терминале

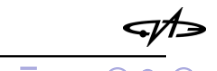
```
> python3
Python 3.6.13 (default, Mar 10 2021, 18:30:35) [GCC] on linux
Type "help", "copyright", "credits" or "license" for more informatic
>>>
```

iPython:

```
Python 3.6.13 (default, Mar 10 2021, 18:30:35) [GCC]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.13.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Веб-интерфейс

```
jupyter notebook
jupyter notebook --ip=127.0.0.1
```



Полезные функции

type() определение типа объекта
dir() список методов объекта
help() краткая справка
len() длина последовательности (строки, списка, множества и др.)

```
type([])
dir(str)
help(str.replace)
len("qwerty")
```

```
help(str.replace)
Help on method_descriptor:

replace(...)
    S.replace(old, new[, count]) -> str

    Return a copy of S with all occurrences of substring
    old replaced by new.  If the optional argument count is
    given, only the first count occurrences are replaced.
```

Ввод данных

Для ввода данных используется функция input():

```
Help on built-in function input in module builtins:

input(prompt=None, /)
    Read a string from standard input.  The trailing newline is
    stripped.

    The prompt string, if given, is printed to standard output
    without a trailing newline before reading input.

    If the user hits EOF (*nix: Ctrl-D, Windows: Ctrl-Z+Return),
    raise EOFError.
    On *nix systems, readline is used if available.
```

```
# Пример
>>> x=input("Введите число:")
Введите число:123
```

Числовые типы

Числовые типы данных Python:

- int
- float
- complex
- bool

```
i = 4
type(i) #  <class 'int'>
f = 4.1
type(f) #  <class 'float'>
c = 4 + 1j
type(c) #  <class 'complex'>
b = (3 > 4)
type(b) # <class 'bool'>
```

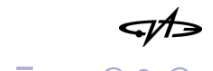
Символьные строки

Строки в Python — неизменяемые последовательности символов.

Строковые литералы:

```
'Это строка'
"И это строка"
""" много
строк """
"Здесь такие кавычки ' не экранируются"
'А здесь - такие " '
```

```
dir(str) # сокращенный вывод
['capitalize','casefold','center','count','encode','endswith',
'expandtabs','find','format','format_map','index','isalnum',
'isalpha','isdecimal','isdigit','isidentifier','islower',
'isnumeric','isprintable','isspace','istitle','isupper','join',
'ljust','lower','lstrip','maketrans','partition','replace',
'rfind','rindex','rjust','rpartition','rsplit','rstrip','split',
'splitlines','startswith','strip','swapcase','title','translate',
'upper', 'zfill']
```



Некоторые методы строк

Метод split() возвращает массив строк, полученных при разбиении исходной строки по заданному разделителю (по умолчанию — пробельные символы)

```
s = "abrakadabra"
s.split("a") # ['', 'br', 'k', 'd', 'br', '']
s2 = "A little cat"
s2.split()  # ['A', 'little', 'cat']
```

Метод join() объединяет строки из заданного массива, используя исходную строку в качестве разделителя

```
a = ["1", "2", "3"]
":".join(a)  # "1:2:3"
```

Метод strip() возвращает строку, в которой удалены пробелы (или заданные символы) в начале и конце строки

```
x = "  spaces are useful "
x.strip()  # "spaces are useful"
s.strip("ar") # "brakadab"
```

Списки

Списки в Python — упорядоченные изменяемые коллекции объектов (допускаются разные типы объектов).
Важно помнить, что списки (как и другие структуры данных — словари, кортежи, множества), содержат ссылки на объекты.

```
L = ['a', 12, [1, 2]]
type(L) # <class 'list'>
ar = [1, 2, 3, 4]
# допустимо ставить запятую после последнего элемента:
ars = ['Sep','Oct','Nov',]
```

```
dir(list)
['append', 'clear', 'copy', 'count', 'extend',
'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Кортежи

Кортежи — неизменяемые структуры данных.

```
t = (1, 3, 17)
t[2]  # 3
t[2] = 4 # ошибка!
s = 'tree'
x = tuple(s) # ('t', 'r', 'e', 'e')
```

Множества

Множество — неупорядоченная последовательность уникальных элементов

```
s = set("abc")
s.union(['a','d']) #  {'a', 'd', 'b', 'c'}
b = {'b','c'}
set("abrakadabra") # {'b', 'r', 'd', 'a', 'k'}
```

```
dir(set)
['add', 'clear', 'copy', 'difference', 'difference_update', 'discard
 'intersection', 'intersection_update', 'isdisjoint', 'issubset',
 'issuperset', 'pop', 'remove', 'symmetric_difference',
  'symmetric_difference_update', 'union', 'update']
```

Словари

```
dict = {'a':1,'b':12,'c':123}
i1 = dict['a'] # результат: i1 = 1
ie = dict['u'] # результат - ошибка KeyError
  # предпочтительнее использовать get:
  # второй аргумент - значение по умолчанию (если ключ не найден)
ie = dict.get('u',42) # результат: ie = 42
```

```
dir(dict)
['clear', 'copy', 'fromkeys', 'get', 'items',
'keys', 'pop', 'popitem', 'setdefault', 'update', 'values']
```

Условный оператор

```
if логическое_выражение1:
    оператор_или_блок
[elif логическое_выражение2:
    оператор_или_блок
...]
[else:
    оператор_или_блок]
```

```
import random
x = random.randint(0, 100)
print(x)
if x > 75:
    print("Up")
elif x > 50:
    print("Middle")
else:
    print("Low")
```

Цикл for

```
for элемент in последовательность:
    оператор_или_блок
```

Последовательностью может быть список, кортеж, словарь, строка, диапазон.
Диапазон (целых чисел) создаётся функцией range() с параметрами.
range(Upper) — от 0 до Upper с шагом 1
range(Start, Upper) — от Start до Upper с шагом 1
range(Start, Upper, Step) — от Start до Upper с шагом Step

```
for i in range(0,10,2):
    print(i)

for x in [3.2, 4.0, 1.23]:
    y = x**2
    print(y)
```

Пользовательские функции

```
def имя_функции(список_параметров):
    тело_функции
```

Необязательная инструкция return позволяет вернуть из функции значение в качестве результата. При отсутствии return в качестве результата возвращается значение None

```
def myfunc(s,n=3):
    return s * n

def myprod(x,f):
    return f(x)
```

Анонимные функции:

```
lambda список_параметров: выражение
```

```
myprod(2, lambda x: x * x)
```

Генераторы списков

```
arr = [1, 2, 3, 4]
a2 = [i*2 for i in arr]
a3 = [i*2 for i in arr if i%2==0]
x = [i**2 for i in range(5)]
```

Если вместо квадратных скобок использовать круглые, будет возвращен объект-итератор, который можно либо перечислить в цикле for, либо создать коллекцию (список, множество, кортеж):

```
xi = (i**2 for i in range(5))
xt = set(xi)
```

Функции map, zip

Функция map применяет функцию, заданную первым аргументом, к каждому элементу заданной последовательности. Результат — объект-итератор (можно его обернуть в вызов list())

```
map(функция, последовательность)

list(map(str,[1, 2, 3])) # результат - список строк ['1', '2', '3']
```

Функция zip возвращает объект-итератор кортежей, составленных из соответственных элементов заданных последовательностей

```
zip(последовательность1, последовательность2)

list(zip([1,2],['a','b'])) # [(1, 'a'), (2, 'b')]
```

Функции filter, reduce

Функция filter применяет предикат, заданный первым аргументом, к каждому элементу заданной последовательности, и помещает в результат только элементы, удовлетворяющие условию отбора. Результат — объект-итератор.

```
filter(функция, последовательность)
filter(None, последовательность)

list(filter(lambda x: x%2==0, [1,2,3])) # [2]
list(filter(None, [0,[],1,False, True])) # [1, True]
```

Функция reduce применяет функцию, являющуюся первым аргументом, к очередному элементу и «накопленному значению»

```
from functools import reduce
reduce(функция, последовательность, начальное_значение)

reduce(lambda x,y:x+y, [1,2,3],0) # 6
```

Индексация списков

Списки поддерживают операцию извлечения среза (подмножества элементов). В общем виде срез задается выражением

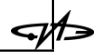
```
[начало:конец:шаг]
```

Любой из трех параметров не является обязательным, для первого параметра значение по умолчанию 0, для второго — конец списка, шаг по умолчанию равен 1. С помощью срезов можно изменять элементы списка.

```
x = [1, 3, 5, 6]
x[-1] # 6 - последний элемент списка
x[1:3] = [] # [1, 6]
```

Срезы применимы и для строк — для получения подмножества символов:

```
s="abrakadabra"
s[1:10:2] # 'baaar'
s[1:-1:2] # 'baaar'
s[::2]    # 'arkdba'
s[::-1]   # 'arbadakarba'
```



Присваивание

В Python реализовано множественное присваивание, позволяющее извлекать отдельные элементы из списков, строк или кортежей. Если количество элементов последовательности больше количества переменных, то для корректного выполнения следует поставить символ '*' перед именем одной из переменных. При этом указанная переменная будет списком, содержащим оставшиеся элементы.

```
a, b, c = [1, 2, 3]
a, b, *c = [1, 2, 3, 4, 5]  # c = [3, 4, 5]
x, *y, z = "abcd" # x='a' y=['b', 'c'] z='d'
```

```
t = tuple([1, 2, 3])
a, b, c = t
a, *b = t # a=1, b=[2,3]
```

zip и словари

```
s="abrakadabra"
sorted(s) # упорядоченный список букв
"".join(sorted(s))  # 'aaaaabbdkrr'
k2 = list(s)
d2 = dict(zip(range(len(k2)),k2))
# {0: 'a', 1: 'b', 2: 'r', 3: 'a', 4: 'k', 5: 'a', 6: 'd',
#  7: 'a', 8: 'b', 9: 'r', 10: 'a'}
d3 = dict(zip(k2,range(len(k2))))
# {'a': 10, 'b': 8, 'r': 9, 'k': 4, 'd': 6}
```

Чтение из файла

```
# создание файлового объекта
infile = open('имя_файла','r')
# чтение всего содержимого файла в одну символьную строку
s = infile.read()
# чтение строки из файла (до символа \n или конца файла)
s = infile.readline()
# чтение всего файла в список строк
arr = infile.readlines()
# итератор строк файла - аналогичен readlines()
for line in infile
```

Запись в файл

```
# создание файлового объекта
outfile = open('имя', 'w')
# Запись строки в файл
outfile.write(S)
# запись в файл все строки из итерируемого объекта
outfile.writelines(I)
```

Пример

```
#!/usr/bin/python3

import sys

if len(sys.argv) < 2:
    fname = "/etc/passwd"
else :
    fname = sys.argv[1]

print(fname)
f = open(fname, "r")
for line in f:
    print(line[:-1])
f.close()
```

Библиотека NumPy работает с массивами (одномерными и многомерными), содержащими элементы одного типа. Тип данных элементов задается при создании массива (явно или неявно).

```
import numpy as np
# неявное задание - по типу элементов
xi = np.array([1, 2, 3, 4])     # целочисленный массив
xx = np.array([1, 2 , 3.5, 4]) # массив чисел с плавающей точкой
xi[0] = 3.14
# тип массива - целочисленный, поэтому значение будет
# автоматически приведено к целому типу, элемент будет равен 3
```

```
# явное задание типа
x1 = np.array([1, 2, 3, 4], dtype='float32')
x2 = np.array([1, 2, 3, 4], dtype=np.float32)
```

Типы данных библиотеки NumPy

Тип данных	Описание
`bool_`	Булев тип (True или False), хранящийся в виде 1 байта
`int_`	Тип целочисленного значения по умолчанию
`intc`	Идентичен типу int языка C (обычно int32 или int64)
`intp`	Целочисленное значение, используемое для индексов
`int8`, `int16`	Целочисленное значение (1 или 2 байта)
`int32`, `int64`	Целочисленное значение (4 или 8 байт)
`uint8`, `uint16`	Беззнаковое целое число (1 или 2 байта)
`uint32`, `uint64`	Беззнаковое целое число (4 или 8 байт)
`float_`	Сокращение для названия типа float64
`float16`	Число с плавающей точкой с половинной точностью
`float32`, `float64`	Число с плавающей точкой с одинарной или двойной точностью
`complex_`	Сокращение для названия типа complex128
`complex64`, `complex128`	Комплексное число, представленное двумя 32-битными или 64-битными числами

Примеры создания массивов NumPy

```
import numpy as np
arr = np.arange(1000) # целые числа от 0 до 999
a2 = np.arange(50,200)# целые числа от 50 до 199
z = np.zeros((3,4)) # двумерный массив, 3 строки, 4 столбца
x = [1, 5, 3]
a = np.array(x)
x = np.linspace(0, 1, 100) # равномерная последовательность
alog = np.logspace(0, 1, 100, base=10.0)
c3 = np.ones((5, 5, 5)).astype(np.float32)
# массив целочисленных нулей
arr = np.zeros(2, dtype=int)
# Массив нулей типа float32
arr = np.zeros(2, dtype=np.float32)
```

Атрибуты и методы массивов NumPy

Атрибуты

ndim	размерность (d)
shape	измерения $(n_1, n_2, \ldots, n_d)$
size	количество элементов $(n_1 \cdot n_2 \cdot \ldots \cdot n_d)$
dtype	тип данных элементов массива
itemsize	размер элемента в байтах
nbytes	размер массива в байтах

Методы

min, max, mean, sum	вычисление минимума, максимума, среднего, суммы
var, std	дисперсия, стандартное отклонение
dot	матричное или скалярное произведение
cumsum, cumprod	кумулятивные сумма и произведение элементов
reshape, resize	изменение размерности (resize изменяет сам массив)

Базовые функции пакета numpy.random

- seed() — задание «затравки» генератора
- rand(), random_sample(), sample(), random(), ranf() — случайные числа из интервала $[0, 1)$
- randn(), standard_normal() — случайные числа, распределенные по стандартному случайному закону
- randint() — целые случайные числа
- choice(), shuffle(), permutation() — случайная выборка из заданного массива или его перемешивание

Примеры:

```
import numpy as np
z1 = np.random.rand(3,4)
z2 = np.random.ranf((10,5))
ir = np.random.randint(-10,10,100)
g1 = np.random.randn(10,5)
g2 = np.random.standard_normal((10,5))
```

Некоторые стандартные распределения

Параметр size (размер получаемого массива) по умолчанию равен 1.

- beta(a,b,size) — бета-распределение
- binomial(n, p, size) — биномиальное распределение
- chisquare(df, size) — распределение χ^2
- exponential(scale, size) — экспоненциальное распределение
- f(df1, df2, size) — распределение Фишера
- gamma(shape, scale, size) — гамма-распределение
- hypergeometric() — гипергеометрическое распределение
- logistic(loc, scale, size) — логистическое распределение
- lognormal(mean, sigma, size) — логнормальное распределение
- normal(loc, scale, size) — распределение Гаусса
- poisson(lam, size) — распределение Пуассона
- power(a, size) — степенное распределение
- standard_t(df, size) — распределение Стьюдента
- triangular(left, mode, right, size) — треугольное распределение
- uniform(low, high, size) — равномерное распределение
- weibull(a, size) — распределение Вейбулла

Индексы и сечения

```
arr = np.zeros((4,5))
# Для получения элемента (0,1):
arr[0,1]
# Для доступа к последнему столбцу:
arr[:,-1]
# Последняя строка:
arr[-1,:]
# Создать массив последовательных целых чисел 0 - 4
arr = np.arange(5)
# Создать массив индексов
index = np.where(arr > 2)
# Получить элементы, не входящие в индекс
new_arr = np.delete(arr, index) # arr не меняется!
#
fr = np.arange(-2.2, 3, 1.1) # [-2.2, -1.1, 0., 1.1, 2.2]
fr[np.where(fr < 1)]         # [-2.2, -1.1, 0.]
```

```
import numpy as np
# Определяем матрицы
A = np.matrix([[3, 6, -5],
               [1, -3, 2],
               [5, -1, 4]])
B = np.matrix([[12],
               [-2],
               [10]])
# Решаем систему, обращая матрицу A
X = A ** (-1) * B
print(X)
# matrix([[ 1.75],
#         [ 1.75],
#         [ 0.75]])
```

```
import numpy as np
a = np.array([[3, 6, -5],
[1, -3, 2],
[5, -1, 4]])
# Определеяем правую часть
b = np.array([12, -2, 10])
# решаем систему обращением матрицы a
x = np.linalg.inv(a).dot(b)
print(x)
# array([ 1.75, 1.75, 0.75])
```

Организация библиотеки SciPy

cluster	алгоритмы кластеризации
constants	физические и математические константы
fftpack	процедуры быстрого преобразования Фурье
integrate	интегрирование и решение ОДУ
interpolate	интерполяция и сплайны
io	ввод-вывод
linalg	линейная алгебра
ndimage	N-dimensional image processing
odr	Orthogonal distance regression
optimize	оптимизация и поиск корней
signal	обработка сигналов
sparse	обработка разреженных матриц
spatial	пространственные алгоритмы и структуры данных
special	специальные функции
stats	статистические распределения и функции

Некоторые методы линейной алгебры

```
Вычисление обратной матрицы:
inv(a, overwrite_a=False, check_finite=True)

Решение системы линейных уравнений
solve(a, b, sym_pos=False, lower=False,
  overwrite_a=False, overwrite_b=False, debug=None,
  check_finite=True, assume_a='gen', transposed=False)

assume_a может принимать значения 'gen', 'sym', 'her', 'pos'

Вычисление определителя матрицы
det(a, overwrite_a=False, check_finite=True)

Вычисление собственных векторов и собственных значений
eig(a, b=None, left=False, right=True, overwrite_a=False,
  overwrite_b=False,check_finite=True,homogeneous_eigvals=False)
```

Линейная алгебра: примеры

```
import numpy as np
from scipy import linalg
a=np.array([[1.,2.1],[2.9,4.1]])
a1=linalg.inv(a) # вычисляем обратную матрицу
print(a1)
print(linalg.det(a)) # вычисляем определитель
print(linalg.eig(a)) # находим с.з. и с.в матрицы
print(linalg.norm(ea[1][:,0])) # норма первого с.в.
print(np.dot(a,a1))  # произведение матриц
```

Результат выполнения:

```
[[-2.0603  1.0553]
 [ 1.4573 -0.5025]]
-1.99
(array([-0.3642+0.j,  5.4642+0.j]), array([[-0.8386, -0.4257],
       [ 0.5448, -0.9049]]))
1.0
[[1. 0.]
 [0. 1.]]
```

Простая реализация метода прямоугольников

```
import numpy as np
def myint(fun,a,b,N):
    x,h = np.linspace(a,b,N,retstep=True)
    return np.sum(fun(x))*h

print(myint(np.sin,0,np.pi,10))
print(myint(lambda x: x**2,0,1,10))
# вывод
1.979650811216483 2
0.390946502057613 1
```

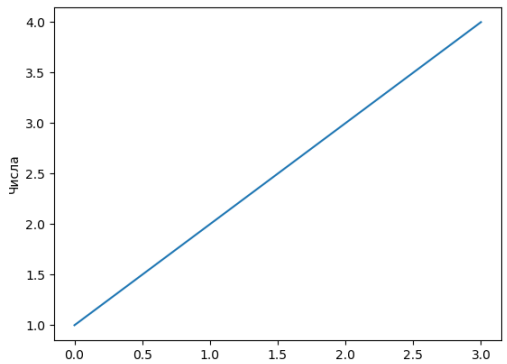
Использование пакета scipy.integrate

```
from scipy import integrate
print(integrate.quad(np.sin,0,np.pi))
print(integrate.quad(lambda x: x**2,0,np.pi))
# вывод: (значение, точность)
(2.0, 2.220446049250313e-14)
(10.335425560099939, 1.1474627426122306e-13)
```

Библиотека matplotlib

matplotlib — библиотека визуализации данных, основанная на использовании массивов NumPy. Простой пример:

```
import matplotlib.pyplot as plt
plt.plot([1, 2, 3, 4])
plt.ylabel('Числа')
plt.show()  # отобразить график
```



Использование библиотеки зависит от контекста:

- выполнение сценария
- интерактивный терминал IPython
- блокнот IPython

Управление визуализацией

Для сценариев следует задать систему поддержки визуализации (backend), например, TkAgg, Qt5Agg, MacOSX, Qt5Cairo и т.д.:

```
import matplotlib
matplotlib.use('Qt5Agg')
```

В интерактивном терминале IPython можно воспользоваться «магической» командой

```
%matplotlib
```

При работе с блокнотами Jupyter «магическая» команда выглядит так:

```
# включение в страницу блокнота статических графиков

%matplotlib inline

```

Структура графика

Библиотека matplotlib поддерживает два варианта интерфейса.
Объектно-ориентированный вариант строится на методах классов, таких, как рисунок и оси координат:
Рисунок (экземпляр класса plt.Figure) — единый контейнер, содержащий систему координат, графику, текст, метки.
Система (оси) координат (экземпляр класса plt.Axes) — ограничивающий прямоугольник с делениями и метками
Второй вариант использует функции пакета matplotlib.pyplot:

```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,10,100)
y = 2*x**2-x
plt.plot(x, y)
plt.plot(x, x**3)
plt.plot(x,np.sin(x))
plt.show() # вызывается один раз!
```

Настройка графика

Параметры графика настраиваются автоматически, используя разумные умолчания.
В то же время многие параметры отображения могут быть настроены пользователем — тип и цвет линий, размеры графика, отображаемые диапазоны изменения, текстовая информация, легенды и т.д.
Для цвета можно выбрать краткий код цвета из набора `[r g b c m y k]`, задать его название в стиле HTML или по RGB-компонентам:

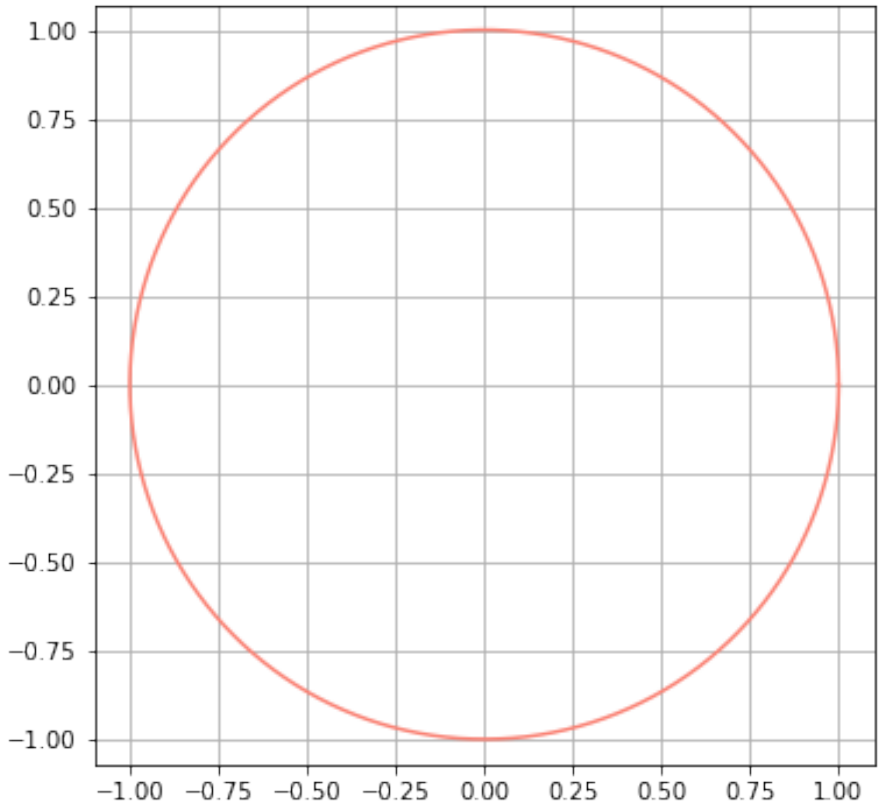
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,10,100)
plt.plot(x, x**2, color = 'r')
plt.plot(x, x**3, color = 'blue')
plt.plot(x,np.sin(x), color = (1.0, 0.1, 0.7))
plt.plot(x, 4-x, color='lightcoral')
plt.plot(x, x, color = '#FFAA00')
plt.show()
```

Вспомогательные функции

```
# задать размер изображения (по умолчанию - в дюймах)
plt.figure(figsize=(dx, dy))
# Задание границ области отображения
plt.axis([xmin, xmax, ymin, ymax])
# Задание диапазонов отображения отдельно для каждой оси
plt.xlim(xmin, xmax)
plt.ylim(ymin, ymax)
# Отключить отображение осей
plt.axis('off')
# Установить одинаковый масштаб по осям
plt.axis('equal')
# Отобразить координатную сетку
plt.grid(True)
# Задать отметки по осям
plt.xticks(xmin, xmax, dx)
plt.yticks(ymin, ymax, dy)
```

Пример

```
import matplotlib.pyplot as plt
import numpy as np
t = np.linspace(0,2*np.pi,1000)
x = np.cos(t)
y = np.sin(t)
plt.figure(figsize=(4, 4))
plt.plot(x, y, color='salmon')
plt.grid(True)
plt.axis('equal')
plt.show()
```



Параметры линий

- linewidth — толщина линии
- linestyle — тип линии, задаётся символьной строкой:
  - `linestyle='-'` — сплошная линия
  - `linestyle='--'` — штриховая линия
  - `linestyle='-.'` — штрихпунктирная линия
  - `linestyle=':'` — пунктирная линия

Указание типа линии может комбинироваться с одиночными символами цвета:

```
plt.plot(x, y, '-r')   # сплошная красная
plt.plot(x, z, '--m')  # штриховая фиолетовая
plt.plot(x, w, ':b')   # пунктирная синяя
plt.plot(x, w, '-.y')  # штрихпунктирная жёлтая
```

Текст на графиках

```
# Текст с заданными координатами
plt.text(x, y, 'Текст')
# Заголовок графика
plt.title('Текст')
# Подпись на оси X
plt.xlabel('Текст')
# Подпись на оси Y
plt.ylabel('Текст')
# Текст с дополнительными параметрами
plt.text(1,2,'Текст',rotation=45,color='red',fontstyle='italic')
```

При выводе текста можно использовать математическую нотацию в стиле LaTeX, используя строки вида r'символы':

```
plt.text(1, 2, r'$\lambda_i = 2\pi$')
plt.ylabel(r'$h=\frac{1}{2}gt^2+v_0 t$')
```

Первое выражение: $\lambda_i = 2\pi$, второе — $h = \frac{1}{2}gt^2 + v_0 t$

Отображение легенды

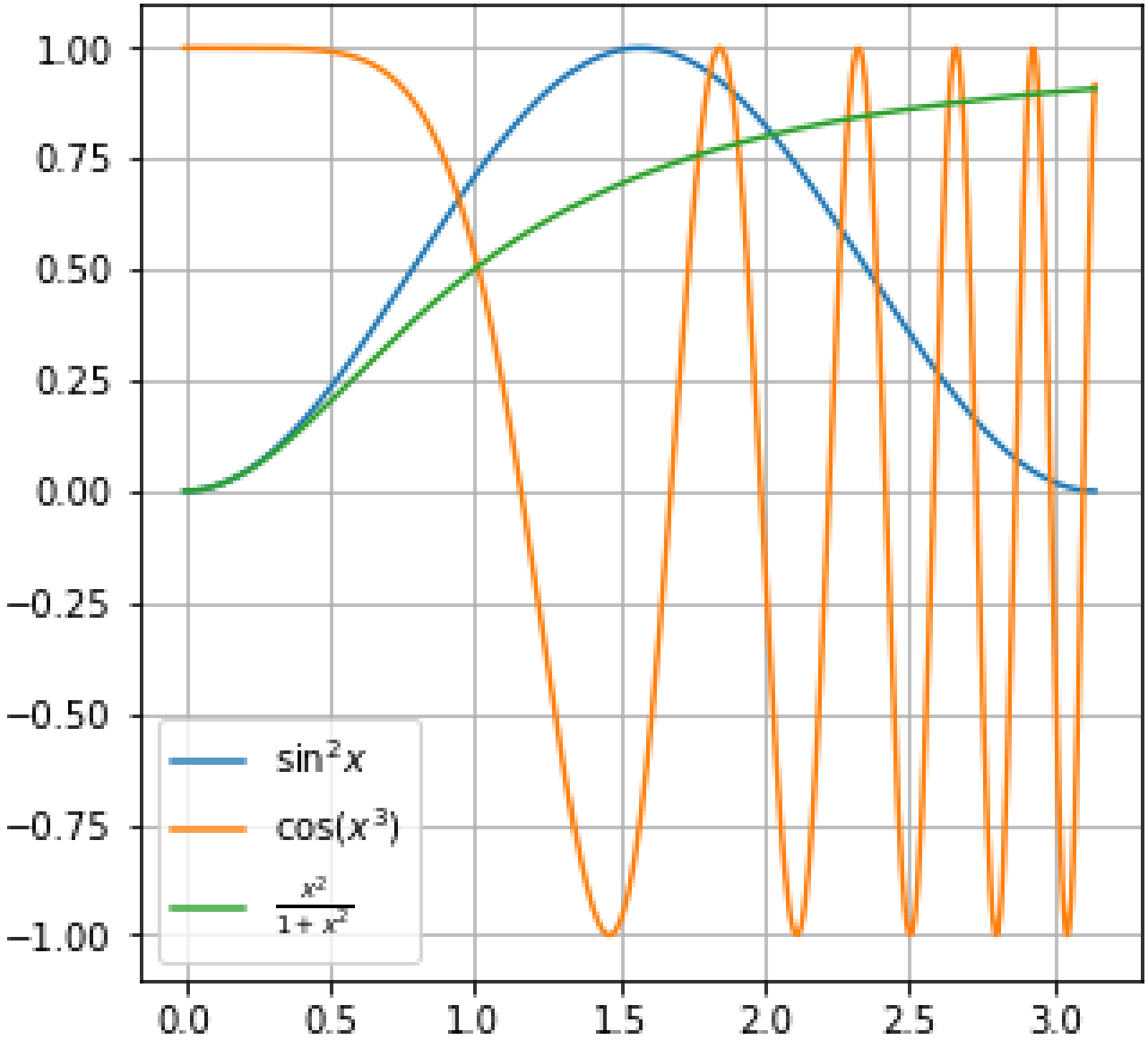
Для отображения легенды следует воспользоваться функцией plt.legend():

```
# Параметр loc='best' задает оптимальное положение легенды
plt.legend(loc='best')
```

Параметр loc может принимать значения 'upper right', 'lower center' и т.д. Если в вызове plt.plot() указан параметр label, то соответствующая строка автоматически попадает в легенду

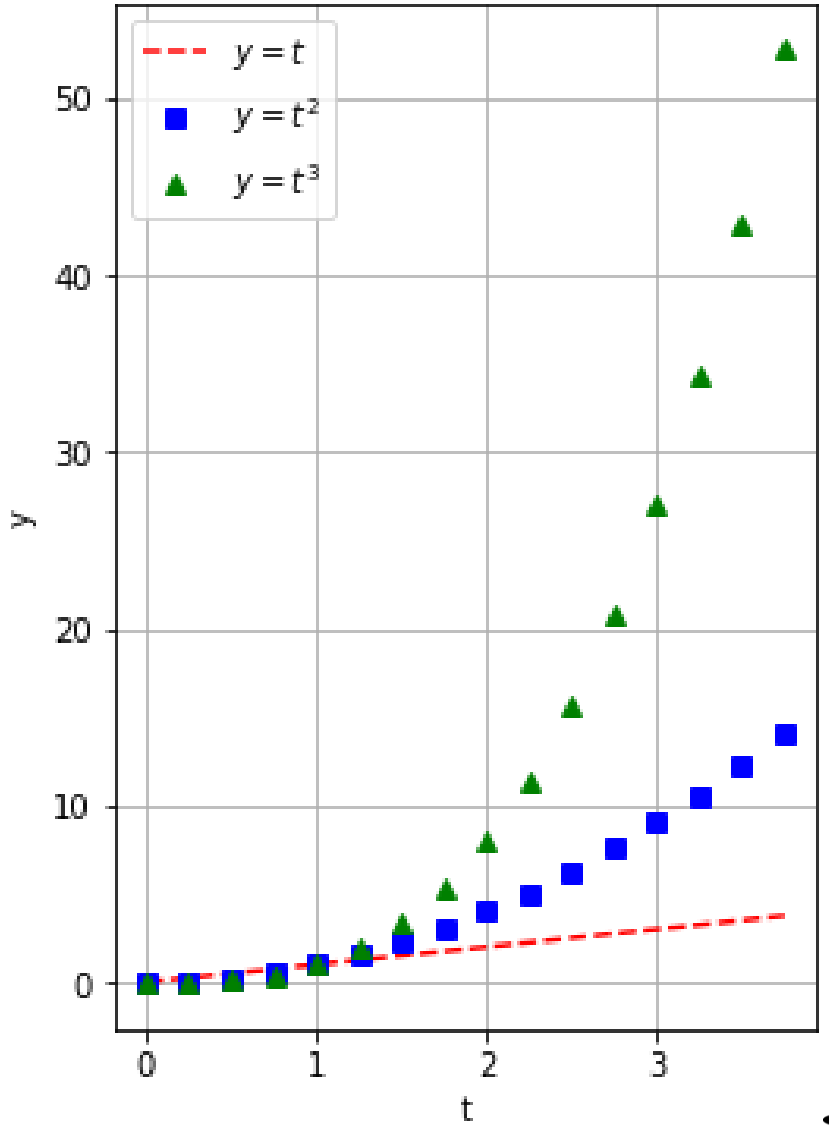
```
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0,np.pi,500)
y = np.sin(x)**2
z = np.cos(x**3)
w = x**2/(1+x**2)
plt.plot(x,y,label=r'$\sin^2 x$')
plt.plot(x,z,label=r'$\cos (x^3$)')
plt.plot(x,w,label=r'$\frac{x^2}{1+x^2}$')
plt.grid(True)
plt.legend()
plt.show()
```

Пример легенды



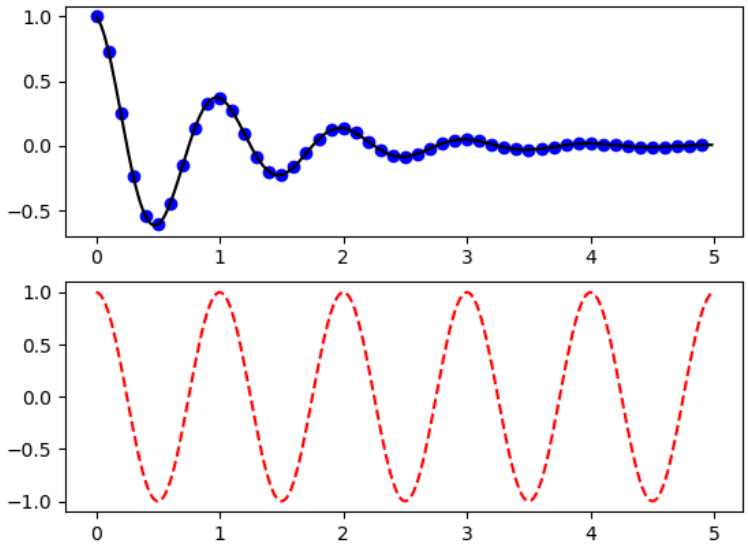
Пример

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

t = np.arange(0., 4., 0.25)
plt.figure(figsize=(4,6))
plt.plot(t, t, 'r--',
         label=r'$y=t$')
plt.plot(t, t**2, 'bs',
         label=r'$y=t^2$')
plt.plot(t, t**3, 'g^',
         label=r'$y=t^3$')
plt.grid(True)
plt.xlabel('t')
plt.ylabel('y')
plt.legend()
plt.show()
```



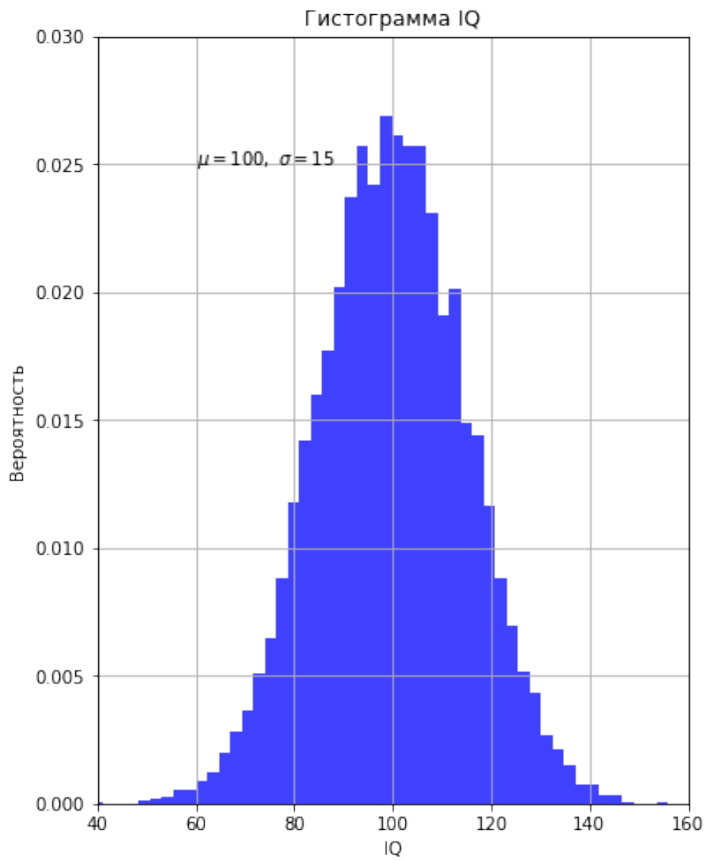
Два графика

```
import matplotlib.pyplot as plt
import numpy as np
def f(t):
  return np.exp(-t)*np.cos(2*np.pi*t)

t1 = np.arange(0.0, 5.0, 0.1)
t2 = np.arange(0.0, 5.0, 0.02)

plt.figure()
plt.subplot(211)
plt.plot(t1, f(t1), 'bo',
         t2, f(t2), 'k')

plt.subplot(212)
plt.plot(t2, np.cos(2*np.pi*t2),
         'r--')
plt.show()
```

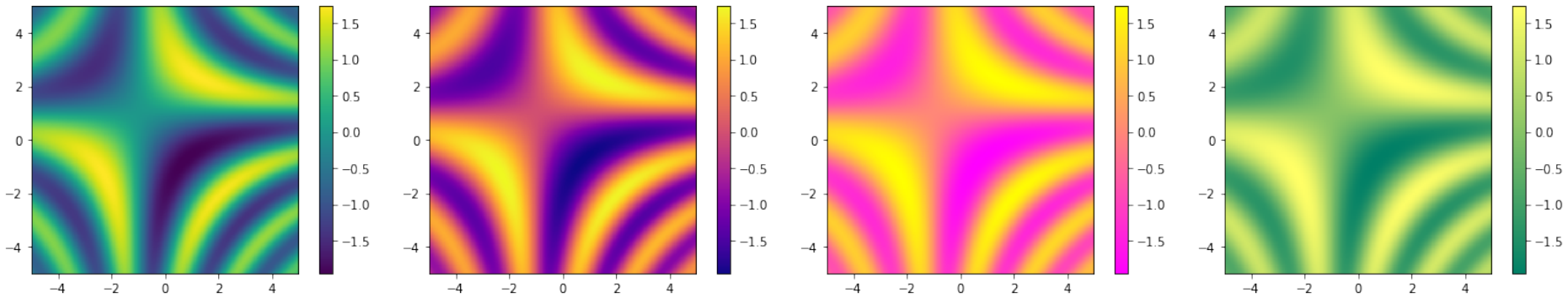


Гистограмма

```
import matplotlib.pyplot as plt
import numpy as np
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)

n, bins, patches = plt.hist(x, 50,
  density=True, facecolor='b',
  alpha=0.75)
plt.xlabel('IQ')
plt.ylabel('Вероятность')
plt.title('Гистограмма IQ')
plt.text(60, .025,
  r'$\mu=100,\ \sigma=15$')
plt.axis([40, 160, 0, 0.03])
plt.grid(True)
plt.show()
```



```
import numpy as np
import matplotlib.pyplot as plt
x=np.linspace(-5,5,400)
y=np.linspace(-5,5,400)[:,np.newaxis]
z=2*np.exp(-0.02*(x**2+y**2))*np.sin(0.5*(x+1)*(y-1))
plt.imshow(z,origin='lower',extent=[-5,5,-5,5],cmap='viridis')
plt.colorbar()
```



`viridis` `plasma` `spring` `summer`

Другие возможные значения параметра cmap: 'inferno', 'magma', 'cividis', 'Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds', 'Spectral', 'coolwarm', 'bwr', 'seismic' (остальные названия можно найти в документации matplotlib)

Решение систем ОДУ

Пакет scipy.integrate содержит функции для решения систем обыкновенных дифференциальных уравнений вида

$$y_i'(t) = f(y_1, \ldots, y_n, t), \qquad i = 1, \ldots, n$$

`scipy.integrate.odeint(func, y0, t,...)`

Здесь

func — функция правой части системы уравнений, должна иметь сигнатуру `func(y, t, ...)`;

y0 — вектор начальных условий;

t — массив значений t, в которых следует найти решение (должен включать начальный момент времени).

```
import scipy.integrate as integrate
```

Решение систем ОДУ

Применим функцию odeint для решения системы Лоренца, задаваемой уравнениями:

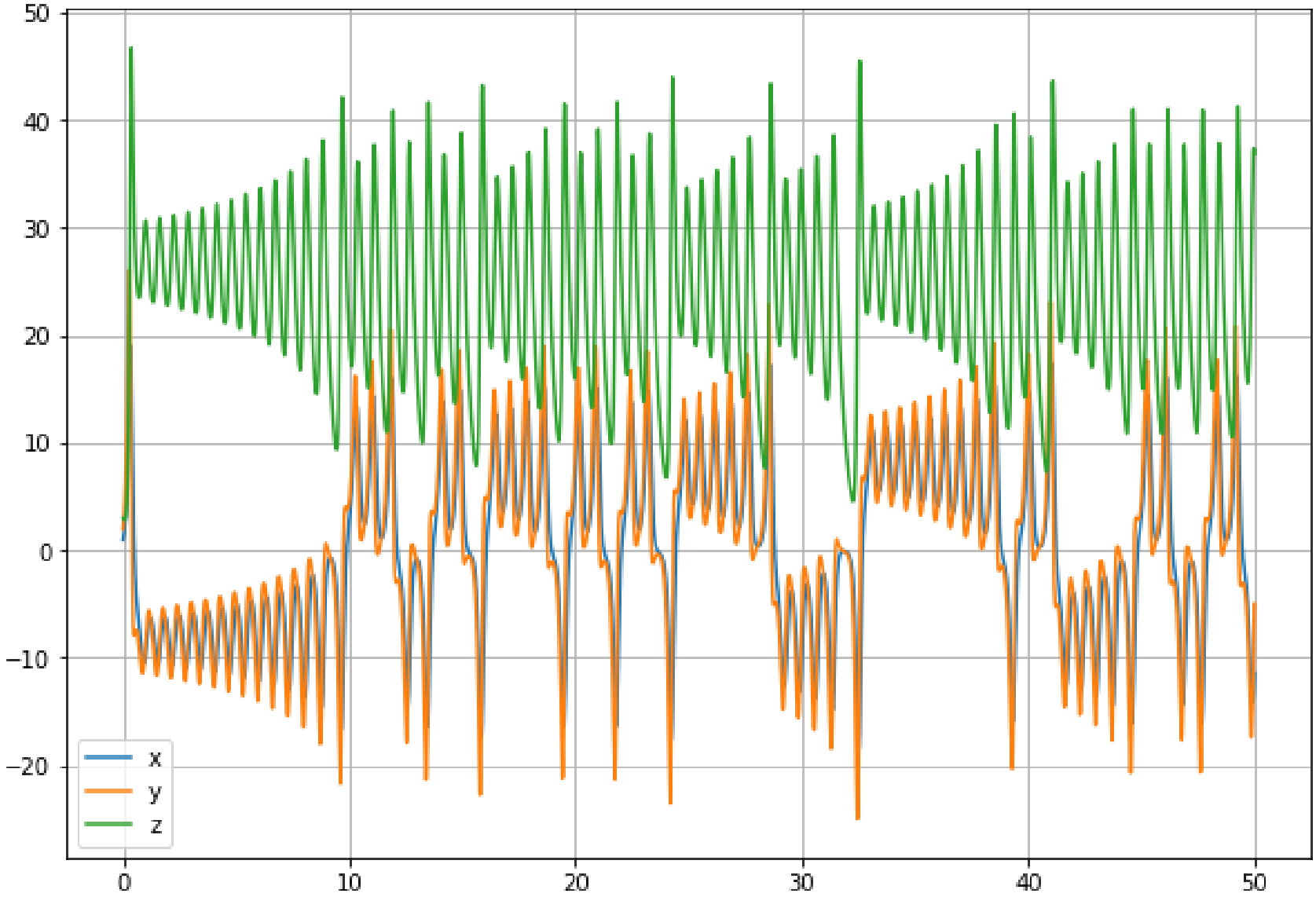
$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = x(r - z) - y \\ \dot{z} = xy - bz \end{cases}$$

Положим $\sigma = 10$, $r = 28$, $b = 8/3$. Тогда функцию правой части и получение решения можно представить как

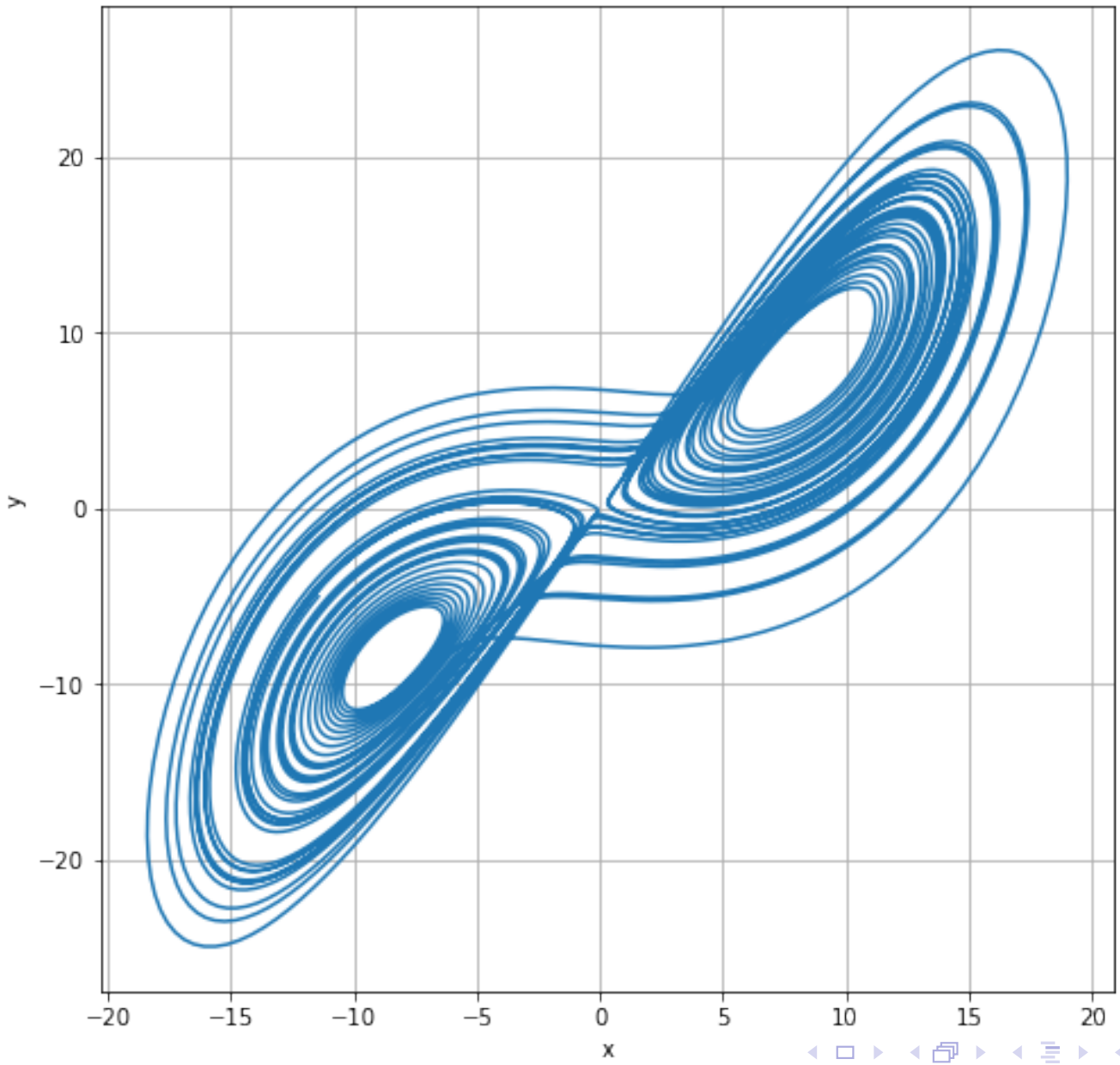
```
import numpy as np
import scipy.integrate as integrate
def lorenz(y,t):
    X,Y,Z=y
    return [10*(Y-X), X*(28-Z)-Y, X*Y-8*Z/3]

def lorsolve(y0,tmax,N=1001):
        t = np.linspace(0,tmax,N)
        return integrate.odeint(lorenz, y0, t)
```

```
import numpy as np
import matplotlib.pyplot as plt
def checklor(flag = True):
    w = lorsolve(y0=[1,2,3], tmax=50, N=10001)
    t = np.linspace(0, 50, 10001)
    x = w[:, 0]
    y = w[:, 1]
    z = w[:, 2]
    plt.figure(figsize=(8,8))
    if flag:
        plt.plot(x,y)
        plt.xlabel('x')
        plt.ylabel('y')
        plt.grid(True)
    else:
        plt.plot(t, x, label='x')
        plt.plot(t, y, label='y')
        plt.plot(t, z, label='z')
        plt.grid(True)
        plt.legend()
    plt.show()
```

Динамика x, y, z



Фазовая плоскость x, y



Пакет knitr

Пакет knitr является развитием пакета Sweave и позволяет с помощью специальных директив разметки включать результаты вычислений и графику R в текстовые документы LaTeX и некоторых других форматов.
Для работы с LaTeX документ следует сохранить в файле с расширением Rnw и провести «конвейер» обработки:

Rnw -> tex -> PDF

В консоли R или в среде Rstudio этот конвейер можно реализовать так:

```
library(knitr)
knit2pdf("имя_файла.Rnw")
```

Директивы разметки

Для включения кода R (и его результатов) в Rnw-файл существует два варианта. Первый (так называемый «чанк»)

```
<<параметры>>=
код на R
@
```

Второй используется для включения результата непосредственно в текст:

```
\Sexpr{код на R}
```

Например:

```
Средний тормозной путь равен \Sexpr{mean(cars$dist)}
```

Пример документа

```
\documentclass{article}
% преамбула от LaTeX
\begin{document}
Текст перед чанком
<<example,fig=TRUE,echo=FALSE>>=
plot(cars)
lm.c = lm(dist ~ speed, data = cars)
abline(lm.c)
@
Текст после чанка
\end{document}
```

Параметры чанка

Флаги (значения TRUE или FALSE, в скобках — значение по умолчанию)

echo — включать команды R в текст документа (TRUE);

eval — выполнять команды R из чанка (TRUE);

warning — включать в код предупреждения, полученные при выполнении кода (TRUE);

error — то же, для сообщений об ошибках (TRUE);

split — вынести код R в отдельные файлы (FALSE);

Параметры чанка

fig.width ширина рисунка (7 дюймов)

fig.height высота рисунка (7 дюймов)

В меню *Options* открыть диалог *Configure TeXstudio*.
Выбрать раздел *Build* и добавить пользовательскую команду, указав следующую последовательность команд:

```
R -e "knitr::knit2pdf('%.Rnw')" |
     txs:///pdflatex |
     txs:///view-pdf-internal
```

Технология R Markdown предоставляет упрощенный (по сравнению с LaTeX) вариант языка разметки для создания документов разных типов и форматов.

Примеры разметки текста:

```
Обычный текст
*курсив* или _курсив_
**жирный** __текст__
верхний индекс^2^
~~зачеркнутый~~

# Заголовок 1

## Заголовок 2
```

Обычный текст
курсив или *курсив*
жирный текст
верхний индекс^2
~~зачеркнутый~~

Заголовок 1

Заголовок 2

```
> цитата в виде
 выделенного абзаца

* список
* элемент списка
  + вложенный список
  + элемент

1. Нумерованный
2. Список

Формула $E=mc^{2}$
```

> цитата в виде
> выделенного абзаца

- список
- элемент списка
  - вложенный список
  - элемент

1. Нумерованный
2. Список

Формула $E = mc^2$

Вставка кода R

Добавление исполняемого кода на R в текст документа производится следующим образом: в отдельной строке записывается ‘‘‘{r} затем идет код на R, завершается блок тремя обратными апострофами на отдельной строке: ‘‘‘

```
Проверим истинность
выражения
$e^{\pi} > \pi^{e}$:
‘‘‘{r}
e=exp(1)
exp(pi) > pi^e
‘‘‘
```

Проверим истинность выражения $e^{\pi} > \pi^{e}$:

```
e=exp(1)
e^pi > pi^e
```

```
## [1] TRUE
```

Для добавления результата непосредственно в текст используется конструкция ‘r код‘.

Установка

- Скачать архив с сайта spark.apache.org
- Распаковать архив
- Перейти в подкаталог распакованного архива и выполнить команду

```
./pyspark
```

Для использования jupyter при работе с Apache Spark следует добавить в файл .bashrc в своем домашнем каталоге следующие команды:

```
# add for spark
export SPARK_HOME=путь_установки_Spark
export PATH=$PATH:$SPARK_HOME/bin:$SPARK_HOME/sbin
export PATH=$PATH:$SPARK_HOME/bin
export PYSPARK_DRIVE_PYTHON="jupyter"
export PYSPARK_DRIVE_PYTHON_OPTS="notebook"
```

и выполнить команду

```
source ~/.bashrc
```

Запуск

После выполнения настройки можно перейти в каталог установки Apache Spark и выполнить команду (в квадратных скобках указывается максимальное количество ядер для вычислений, символ * означает использовать все доступные)

```
./bin/pyspark --master local[*]
```

При этом запустится веб-интерфейс jupyter.

```
[I 11:08:50.488 NotebookApp] Loading IPython parallel extension
[I 11:08:50.489 NotebookApp] Serving notebooks from local
 directory: /home/oam/spark/spark-3.1.1-bin-hadoop2.7
[I 11:08:50.489 NotebookApp] The Jupyter Notebook is running at:
[I 11:08:50.489 NotebookApp]
 http://localhost:8888/?token=e4162cbe9e51a9298c3649b704778a5dc41
[I 11:08:50.489 NotebookApp] Use Control-C to stop this server
 and shut down all kernels (twice to skip confirmation).
```

Выполнение

```
from pyspark import SparkContext
sc = SparkContext(master = 'local[2]')
print(sc.version)
print(sc.pythonVer)
print(sc.master)
print(str(sc.sparkHome))
print(str(sc.sparkUser()))
print(sc.appName)
```

Конфигурирование

```
from pyspark import SparkConf, SparkContext
conf = (SparkConf().setMaster("local)
                   .setAppName('MyApp)
                   .set("spark.executor.memory", "1g"))
sc = SparkContext(conf = conf)
```

Загрузка файла

```
tf = sc.textFile("/path/to/file.txt")
```

Чтение данных

Базовые функции R для чтения данных из файлов:

```
read.table(file,header=FALSE,sep="",dec=".",...)
read.csv(file,header=TRUE,sep=",",dec = ".",...)
read.csv2(file,header=TRUE,sep=";",dec = ",",...)
read.delim(file,header=TRUE,sep="\t",dec = ".",...)
read.delim2(file,header=TRUE,sep="\t",dec = ",",...)
```

Параметр file обычно — имя файла

Для измерения времени выполнения операции в R

```
stt = proc.time()
код для выполнения
proc.time() - stt
```

Модификации data.frame

Сравнение скорости чтения:

```
 ptm = proc.time()
 x = read.table("ch1_5",head=F)
 proc.time() - ptm
-----------------------------------
пользователь  система     прошло
50.284        1.116       51.381
```

```
 ptm = proc.time()
 y = fread("ch1_5")
 proc.time() - ptm
-----------------------------------
Read 2696195 rows and 19 (of 19) columns
from 0.512 GB file in 00:00:08
пользователь      система       прошло
7.596         0.180         7.770
```

```
> di = tibble(iris)
> di
# A tibble: 150 x 5
Sepal.Length Sepal.Width Petal.Length Petal.Width
<dbl>        <dbl>        <dbl>        <dbl>
1            5.1          3.5          1.4          0.2
2            4.9          3.0          1.4          0.2
3            4.7          3.2          1.3          0.2
4            4.6          3.1          1.5          0.2
  ...
9            4.4          2.9          1.4          0.2
10           4.9          3.1          1.5          0.1
# ... with 140 more rows, and 1 more variables:
#   Species <fctr>
```

dplyr: операции с таблицами

```
glimpse()     # информация о таблице
filter()      # отбор строк по условию
select()      # отбор столбцов
arrange()     # сортировка строк
rename()      # переименование столбцов
distinct()    # уникальные строки
sample_n()    # случайная выборка заданного числа строк
sample_frac()# случайная выборка заданной доли строк
slice()       # выборка по номерам строк
group_by()    # группировка строк
mutate()      # изменение структуры таблицы
summarise()   # агрегация значений
```

Примеры

```
dx = tibble(iris)
dy = select(dx,Sepal.Length,Sepal.Width,Species)
dz = filter(dy,Sepal.Length > 5.2)
dw = rename(dz,sl=Sepal.Length, sw=Sepal.Width)
dq = mutate(dw,sdiv=sl/sw)
dr = arrange(dq,desc(sl))
dg = group_by(dr,Species)
ds = summarise(dg,mean(sl),max(sw),sum(sdiv),n())
```

Результат

```
summarise(dg,mean(sl),max(sw),sum(sdiv),n())
# A tibble: 3 x 5
Species `mean(sl)` `max(sw)` `sum(sdiv)` `n()`
<fctr>      <dbl>     <dbl>       <dbl> <int>
1     setosa   5.500000       4.4     15.9640    11
2 versicolor   6.035556       3.4     97.3386    45
3  virginica   6.622449       3.8    109.5626    49
```

Неудобство — в большом количестве промежуточных результатов.

Оператор %>%

Пакет dplyr с помощью оператора %>%, введенного в пакете magrittr, позволяет организовывать «конвейер» или цепочку операций, когда результат вызова одной функции передается как первый аргумент следующей функции. Приведенный ранее пример можно записать следующим образом:

```
res = tibble(iris) %>%
      select(Sepal.Length,Sepal.Width,Species) %>%
      filter(Sepal.Length > 5.2) %>%
      rename(sl=Sepal.Length, sw=Sepal.Width) %>%
      mutate(sdiv=sl/sw) %>%
      arrange(desc(sl)) %>%
      group_by(Species) %>%
      summarise(mean(sl),max(sw),sum(sdiv),n())
```

Функции для агрегации

Функция summarise() может использовать следующие вспомогательные функции при агрегации сгруппированных данных:

```
# из пакета dplyr
  first()
  last()
  nth()
  n()
  n_distinct()
```

```
# стандартные
  IQR()
  mean()
  median()
  min()
  max()
  sd()
  var()
```

Чтение данных

Базовые функции R для чтения данных из файлов:

```
spark_read_csv(sc, name, path, header=TRUE,...)
spark_read_json(sc, name, path, options=list(),...)
spark_read_parquet(sc, name, path, options=list(),...)
```

Операции dplyr

```
select ~ SELECT
filter ~ WHERE
arrange ~ ORDER
summarise ~ aggregators: sum, min, sd, etc.
mutate ~ operators: +, *, log, etc
```

«Ленивый» доступ

При работе с внешними источниками данных dplyr старается использовать «ленивый» доступ:

- данные не загружаются в R без явного указания;
- выполнение действий откладывается до последнего момента: параметры требуемых последовательных операций собираются и затем отправляются источнику данных.

Для загрузки результата в R используется функция `collect()`

Пример

```
ci=copy_to(sc,iris)
li=ci %>% select(Sepal_Length,Sepal_Width,Species) %>%
             filter(Sepal_Length > 4.7) %>%
             group_by(Species) %>%
             summarise(n=n(),sl = mean(Sepal_Length),
                             sw = mean(Sepal_Width))
collect(li)
```

```
 # A tibble: 3 x 4
  Species        n    sl    sw
  <chr>      <dbl> <dbl> <dbl>
1 versicolor    50  5.94  2.77
2 virginica     50  6.59  2.97
3 setosa        39  5.14  3.52
```

Обзор функций sparklyr для ML

Пакет sparklyr предоставляет функции для доступа к библиотеке алгоритмов машинного обучения MLlib Spark.

- функции для работы со Spark DataFrames используют в названии префикс `sdf_`
- имена методов преобразования данных начинаются с `ft_`
- в названиях алгоритмов машинного обучения используется префикс `ml_`