Алгоритмы обработки данных с МЭМС-датчиков в ИС

Введение в язык описания сценариев Python

Соловьев А. В.

ПетрГУ – КИИСиФЭ

(Rev. 2016 07 09)

Почему Python?

Достоинства:

- Большая коллекция вычислительных библиотек (NumPy, SciPy, Pandas, Matplotlib, ...)
- Скриптовый язык (не требуется компиляция, распределение памяти и т. п.)
- Универсальный язык – «для всего» (по ср. PHP)
- Бесплатный и свободный
- Достаточно быстрый среди скриптовых языков (медленнее Perl, но быстрее Java)
- Красивый синтаксис (дело вкуса)

Недостатки:

- Уступает по функциональным возможностям Matlab
- Среда разработки не столь комфортна (дело вкуса)

Интерпретаторы, среда исполнения

Средства разработки:

- Кроме базового интерпретатора python, используется IPython – расширенный интерпретатор (история команд, мониторинг переменных, контекстная информация об объектах и др.)
- Библиотеки: NumPy, SciPy, matplotlib, Mayavi, Pandas, ...
- Текстовый редактор: блокнот, SciTE, ...
- Среда разработки: IDLE, SPYDER, ...

Дистрибьютивы Python (для Win):

- Anaconda (www.continuum.io)
- WinPython (winpython.github.io)
- Python(x,y) (python-xy.github.io)

Синтаксис Python

Основные типы

int	`100, 0x100, 0100, 0b100`
float	`3.14`
complex	`1.5+0.5j`
bool	`True, False`

Переменные не объявляются, тип может меняться в процессе работы.

Преобразование типов:

`float(1)`

Основные арифметические операции:

`+ - * / % ** //`

Сложные типы: списки (1)

```
>>> a = ['apple', 1, 3.14, 0.5j]
>>> type(a)
<type 'list'>
>>> a[0]
'apple'
>>> a[-1]
0.5j
>>> a[1:3]
[1, 3.14]
>>> a[::-1]
[0.5j, 3.14, 1, 'apple']
>>> a[1] = 0
>>> a
['apple', 0, 3.14, 0.5j]
```

- Элементы разных типов
- Индексы от нуля, отрицательные – с конца
- Фрагменты списка
- Элементы изменяемые
- Функции: len, sorted,...

Сложные типы: списки (2)

Списки являются объектами. Некоторые операции для них перегружены:

- \+ – конкатенация:

```
>>> a + a[::-1]
['apple', 0, 3.14, 0.5j, 0.5j, 3.14, 0, 'apple']
```

- * – повторение

```
>>> a * 2
['apple', 0, 3.14, 0.5j, 'apple', 0, 3.14, 0.5j]
```

Методы списков:

- append() – добавление элемента в список,
- pop() – извлечение последнего элемента из списка,
- reverse() – перестановка в обратном порядке,
- sort() – сортировка,
- и др.

Сложные типы: строки (1)

```
>>> s = 'Hello'
>>> s[2:]
'llo'
```

Строковые константы заключаются в кавычки одинарные или двойные. Многострочные строковые константы заключаются в тройные кавычки (три пары ' или "). Символы строки индексируются так же, как элементы списков.

```
>>> print 'c:\new\folder'
c:
ew
  older
>>> print r'c:\new\folder'
c:\new\folder
```

Строковые константы, начинающиеся с литеры r ("raw"), могут содержать любую последовательность кодов (отменяется специальное значение обратного слеша)

Сложные типы: строки (2)

Строки – неизменяемый объект!!!

```
>>> s[2]='z'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'str' object does not support item assignment
>>> s.replace('l','z')
'Hezzo'
>>> s
'Hello'
```

Операция % – форматирование в стиле *printf*:

```
>>> q='%d %f' % (a[1],a[2])
>>> print q
1 3.140000
```

Сложные типы: словари

```
>>> d = {'a':1, 'b':2, 3:'hello'}
>>> d
{'a': 1, 3: 'hello', 'b': 2}
>>> d['a'] = 0.5j
>>> d
{'a': 0.5j, 3: 'hello', 'b': 2}
>>> d['new'] = 5
>>> d
{'a': 0.5j, 3: 'hello', 'b': 2, 'new': 5}
```

Ключи и значения могут быть разных типов. Элементы изменяемые.

Сложные типы: кортежи (tuples), множества (sets)

Кортежи ведут себя как неизменяемые списки. Элементы записываются в круглых скобках или вообще без скобок:

```
>>> t = 12345, 54321, 'hello!'
>>> t[0]
12345
>>> t
(12345, 54321, 'hello!')
>>> u = (0, 2)
```

Множества содержат неупорядоченные уникальные элементы:

```
>>> s = set(('a', 'b', 'c', 'a'))
>>> s
set(['a', 'c', 'b'])
>>> s.difference(('a', 'b'))
set(['c'])
```

Управляющие конструкции: if/elif/else (1)

Составной оператор начинается после двоеточия. Блок операторов следует с отступом.

```
>>> if a == 1:
...     print(1)
... elif a == 2:
...     print(2)
... else:
...     print('A lot')
```

Условие считается ложным, если имеет численное значение 0 (0.0 или 0+0j), если имеет логическое значение False (или None), если является пустым контейнером (списком, кортежем, множеством, словарём, строкой).

Управляющие конструкции: if/elif/else (2)

Особенности некоторых логических операций:

```
>>> a=[1,2,3]
>>> b=[1,2,3]
>>> a == b
True
>>> a is b
False
>>> c=a                          # c -- это ссылка на a
>>> c is a
True
>>> 1 in b
True
>>> 5 in b
False
```

Управляющие конструкции: for/range

```
>>> for i in range(4):
...     print(i)
0
1
2
3
```

range(*stop*),
range(*start*,*stop*),
range(*start*,*stop*,*step*)
— генерирует список

```
>>> for word in ('black', 'red', 'green', 'blue'):
...     print('%s tea' % word)
black tea
red tea
green tea
blue tea
```

Управляющие конструкции: while/break/continue

```
>>> z = 1+1j
>>> while abs(z)<100:
...     if z.imag == 0:
...         break
...     z = z**2 + 1
>>> z
(-134+352j)
>>> a = [1, 0, 2, 4]
>>> for i in a:
...     if i == 0:
...         continue
...     print(1. / i)
1.0
0.5
0.25
```

Управляющие конструкции: pass

Оператор pass не делает ничего. Используется, если по синтаксису в данном месте программы нужен какой-то оператор, но по логике действий не требуется.

```
>>> while True:
...     pass                                # Пустой цикл

>>> class EmptyClass:
...     pass                                # Пустой класс

>>> def myfunc(*args):
...     pass                                # Пустая функция
```

Управляющие конструкции: else

Циклы могут содержать опеатор `else`. Этот блок исполняется, если исчерпан список (для `for`) или условие стало ложным (для `while`). Этот блок не исполняется, если цикл прерван `break`.

```
>>> for n in range(2, 10):
...     for x in range(2, n):
...         if n % x == 0:
...             print(n, 'equals', x, '*', n//x)
...             break
...     else:
...         print(n, 'is a prime number')
```

Управляющие конструкции: прочее

Подстановка списков:

```
>>> import math
>>> y = [math.sin(math.pi*i/100.) for i in range(100)]
```

Итерация словарей:

```
>>> d = {'a': 1, 'b':1.2, 'c':1j}
>>> for key, val in sorted(d.items()):
...     print('Key: %s has value: %s' % (key, val))
Key: a has value: 1
Key: b has value: 1.2
Key: c has value: 1j
```

Функции: особенности

- При описании функции (оператор `def`) используются отступы!
- Если функция возвращает значение, используется оператор `return`.
- Параметры могут иметь значения по умолчанию (и пропускаться).
- При вызове функции можно использовать имена параметров (ключи) в произвольном порядке.

Функции: пример

```
>>> def slicer(seq, start=None, stop=None, step=None):
...     return seq[start:stop:step]
>>> a = [1, 2, 3, 4, 5, 6, 7, 8]
>>> slicer(a)
[1, 2, 3, 4, 5, 6, 7, 8]
>>> slicer(a,1,step=2)
[2, 4, 6, 8]
>>> slicer(a, step=2, start=1, stop=4)
[2, 4]
```

Функции: глобальные переменные

```
>>> x=1
>>> def setx(y):
...     x = y
...     print('x is %d' % x)
>>> setx(10)
x is 10
>>> x
1
>>> def setx(y):
...     global x
...     x = y
...     print('x is %d' % x)
>>> setx(10)
x is 10
>>> x
10
```

Внутри функции значения глобальных переменных доступны только для чтения, если переменная не объявлена как `global`.

Функции с переменным числом аргументов

Две специальные формы переменного числа параметров:

- аргументы обрабатываются в виде кортежа:

```
>>> def test(*args):
...     print 'args are', args
>>> test(1,'apple',3.14)
args are (1, 'apple', 3.14)
```

- аргументы обрабатываются в виде словаря:

```
>>> def test(**args):
...     print 'args are', args
>>> test(a=1,s='apple',pi=3.14)
args are {'a': 1, 's': 'apple', 'pi': 3.14}
```

Документирование кода

Строковая константа в самом начале блока объявления функции, класса, метода или модуля является строкой описания (документацией) на данный объект.

```
>>> def sum(x,y):
...     """Add x to y and return te result."""
...     return x+y
>>> sum(3,5)
>>> help(sum)
Help on function sum in module __main__:

sum(x, y)
    Add x to y and return te result.
```

Запуск скриптов

Скрипт (текстовый файл с последовательностью инструкций на Python) можно запустить из приглашения интерпретатора (`%run` в IPython или `execfile` в эталонном интерпретаторе python) либо из командного интерпретатора ОС, указав запускаемый скрипт как параметр команды python.

```
$ python
Python 2.7.9 (default, Mar  1 2015, 18:22:53)
>>> execfile('test.py')

$ ipython
Python 2.7.9 (default, Mar  1 2015, 18:22:53)
In [1]: %run test.py

$ python test.py
```

Модули: импортирование объектов/функций

```
>>> import math
>>> math.cos(0)
1.0
```

Модули – внешние библиотеки объектов/функций. Перед использованием их необходимо подключить – «импортировать». Идентификаторы объектов/функций указываются с обозначением модуля, в котором они объявлены.

```
>>> import matplotlib as mpl
>>> mpl.pyplot.plot(range(10))
>>> mpl.pyplot.show()
```

```
>>> from matplotlib import pyplot as plt
>>> plt.plot(range(10))
>>> plt.show()
```

Можно указать короткий идентификатор, под которым модуль или объект из него будет использоваться.

Ввод/вывод текстовых файлов

```
>>> f = open('filename.txt', 'r')     # открыть файл для чтения
>>> str = f.readline()                # прочитать 1 строку
>>> str = f.read(128)                 # прочитать 128 байт
>>> str = f.read()                    # прочитать всё
>>> f.close()
>>> f = open('/etc/passwd', 'r')
>>> for line in f:                    # итерация по файлу
...     print line                    # даёт очередную строку
>>> f.close()
```

При достижении конца файла f.read() возвращает пустую строку.

```
>>> f = open('filename.txt', 'w')     # открыть файл для записи
>>> f.write('Some text')
>>> f.close()
```

Стандартные библиотеки (модули)

- os – различные интерфейсы операционной системы (атрибуты процесса, манипуляции файловой системой и т.п.)
- math – математические функции (степени, корни, логарифмы, тригонометрические и проч. функции)
- cmath – математические функции с комплексными аргументами
- re – регулярные выражения
- io – потоковый ввод-вывод
- sys – параметры и функции, специфичные для платформы (аргументы командной строки, порядок байт, версия, разрядность, платформа и т. п.)

Исключения (1)

```
>>> 1/0
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero

>>> while True:
...     try:
...         x=int(raw_input("Please enter a number:"))
...         break
...     except ValueError:
...         print "Oops! Try again..."
...
Please enter a number:s
Oops! Try again...
Please enter a number:1
>>> x
1
```

Исключения (2)

`try: ... except: ... else: ... finally: ...`

Сначала исполняется блок `try`.

Если исключений не возникло, блоки `except` пропускаются, исполняется блок `else` (может отсутствовать).

Если при исполнении блока `try` возникает исключение, остальные операторы этого блока пропускаются. Возникшее исключение сопоставляется с параметрами блоков `except`.

Если совпадение найдено, выполняется соответствующий блок.

Если совпадений не найдено, то это *неперехваченное исключение*, оно вызывает остановку исполнения скрипта и вывод контекста исключения на экран.

Блок `finally` исполняется в любом случае.

Вариант синтаксиса блока `except`:

```
except (RuntimeError, TypeError, NameError) as ex:
```

Объектно-ориентированный подход

```
>>> class Student(object):
...     """Docstring"""
...     institution = "PetrSU"
...     def __init__(self, name):
...         self.name = name
...     def set_age(self, age):
...         self.age = age
>>> anna = Student('Anna')
>>> anna.set_age(21)
>>> Student.institution
PetrSU
```

- множественное наследование
- нет сокрытия (всё public)
- методы всегда виртуальные
- первый параметр – экземпляр, если метод нестатический

Библиотека NumPy

Массивы NumPy – общие сведения

Массивы NumPy состоят из элементов одного типа – float, complex, bool, string, int.
Это позволяет сэкономить память и время:

```
In [1]: L=range(1000)
In [2]: %timeit [i**2 for i in L]
10000 loops, best of 3: 98.2 µs per loop
In [3]: import numpy as np
In [4]: a = np.arange(1000)
In [5]: %timeit a**2
100000 loops, best of 3: 4.39 µs per loop
```

Массивы NumPy – создание

Пустой (с мусором в памяти):

```
e = np.empty(10)                          # на 10 элементов
```

Из списка:

```
a = np.array([[0, 1, 2], [3, 4, 5]]) # 2 строки x 3 столбца
```

Периодические отсчёты:

```
x = np.arange(0, np.pi, 0.01)         # 0...3.14 с шагом 0.01
y = np.linspace(0, 1, 6)              # 0...1, всего 6 точек
```

Специальные:

```
a = np.ones((3,3))                    # из единичек 3x3
b = np.zeros(10)                      # из нулей, всего 10 шт.
c = np.eye(3)                         # диагональная матрица 3x3
```

Со случайными элементами:

```
a = np.random.rand(3,4)               # равномерно из [0,1]: 3x4
b = np.random.randn(5)                # нормальное N(0,1): 5 шт.
```

Массивы NumPy – индексирование

Индексирование и фрагменты – как у обычных списков.

```
>>> a=np.random.rand(10)
>>> a
array([ 0.0328,  0.8817,  0.3465,  0.9440,  0.7902,
        0.3544,  0.1674,  0.2598,  0.4791,  0.0073])
>>> a[::-1]
array([ 0.0073,  0.4791,  0.2598,  0.1674,  0.3544,
        0.7902,  0.9440,  0.3465,  0.8817,  0.0328])
```

Массивы NumPy допускают использование в качестве индекса массивов.

```
>>> a[[1,3,5]]
array([ 0.8817,  0.9440,  0.3544])
```

Для многомерных массивов в качестве индекса используется кортеж.

Операции над массивами

Основные арифметические операции – поэлементные:

```
>>> np.ones(5)+1
array([ 2.,  2.,  2.,  2.,  2.])
>>> np.array([1,2,3])*np.array([1,2,3])
array([1, 4, 9])
```

Для математических функций есть специальные реализации:

```
x = np.arange(0, np.pi, 0.01)
y = np.sin(x)+np.cos(2*x)
```

Матричное перемножение:

```
c = np.dot(a,b)
```

Транспонирование матрицы (даёт view):

```
x.T
```

Полиномы (1)

В NumPy есть два интерфейса для работы с полиномами: numpy.poly1d и numpy.polynomial.
$3x^2 + 2x - 1$:

```
>>> import numpy as np
>>> p1=np.poly1d([3,2,-1])
>>> print p1
   2
3 x + 2 x - 1

>>> p2=np.polynomial.Polynomial([-1,2,3])
>>> print p2
poly([-1.  2.  3.])
```

Полиномы (2)

Корни:

```
>>> p1.roots
array([-1.        ,  0.33333333])
>>> p2.roots()
array([-1.        ,  0.33333333])
```

Степень полинома:

```
>>> p1.order
2
>>> p2.degree()
2
```

Коэффициенты:

```
>>> p1.coeffs
array([ 3,  2, -1])
>>> p2.coef
array([-1.,  2.,  3.])
```

Операции с полиномами

```
>>> print p
   2
3 x + 2 x - 1
>>> p(0)
-1
>>> print p+5
   2
3 x + 2 x + 4
```

```
>>> print p*2
   2
6 x + 4 x - 2
>>> print p**2
   4      3     2
9 x + 12 x - 2 x - 4 x + 1
>>> print p(p)
    4      3
27 x + 36 x - 8 x
```

Загрузка данных

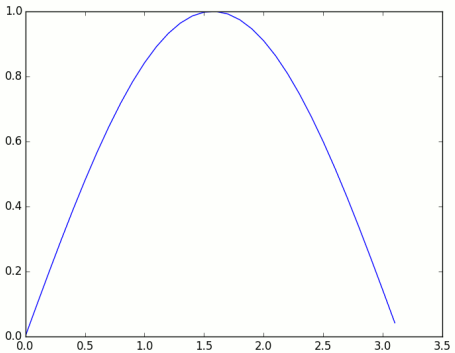
```
Example: populations.txt:
# year hare    lynx    wolf
1900   30e3    4e3     48300
1901   47.2e3  6.1e3   48200
1902   70.2e3  9.8e3   41500

>>> data = np.loadtxt('data/populations.txt')
>>> data
array([[ 1900., 30000., 4000., 48300.],
       [ 1901., 47200., 6100., 48200.],
       [ 1902., 70200., 9800., 41500.]])
```

Matplotlib: построение графиков

Простые графики

```
>>> from matplotlib import pyplot as plt;
>>> x=np.arange(0,3.14,0.1)
>>> y=np.sin(x)
>>> plt.plot(x,y)
[<matplotlib.lines.Line2D object at 0xb0768b0c>]
>>> plt.show()
```

В зависимости от режима работы интерпретатора вызов show() для окна графика может требоваться или нет.
Если используется ipython notebook, в начало заметки надо вставить:
`%matplotlib inline`



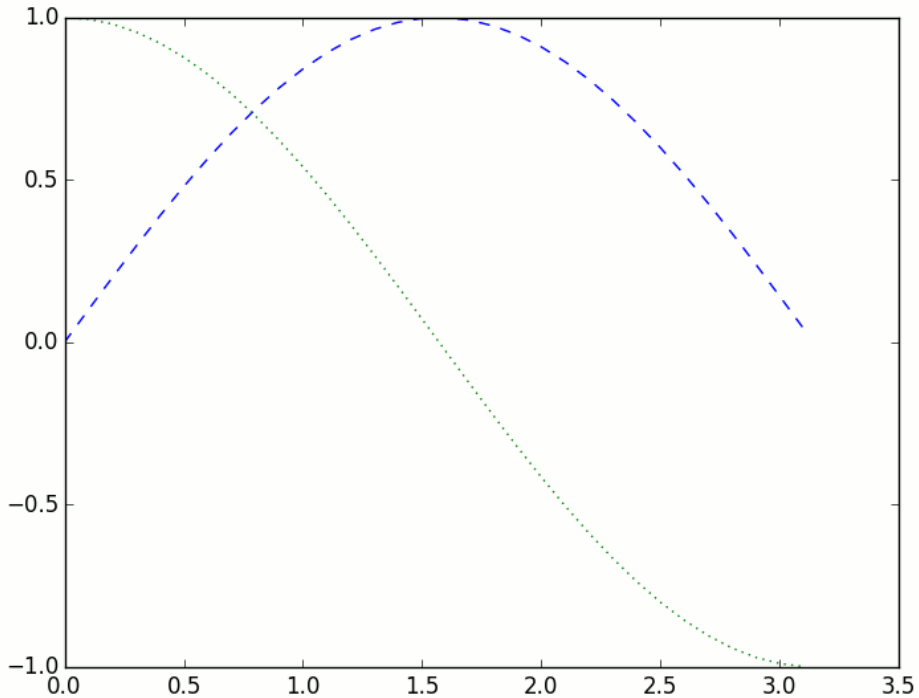
Атрибуты графика

- color – цвет: 'c' (сине-голубой), 'b' (синий), 'w' (белый), 'g' (зелёный), 'y' (жёлтый), 'k' (чёрный), 'r' (красный), 'm' (фиолетовый)
- linestyle – стиль линии: '-' (сплошная), '–' (пунктирная), '-.' (штрих-пунктирная), ':' (точечная) и др.
- linewidth – ширина линии в точках (float)
- marker – обозначение отсчётов: '+', '.', 'o', '1', '2', ...

```
>>> plt.plot(x,y,color='k',linestyle='--',linewidth=2.0)
```

Атрибуты графика (пример 1)

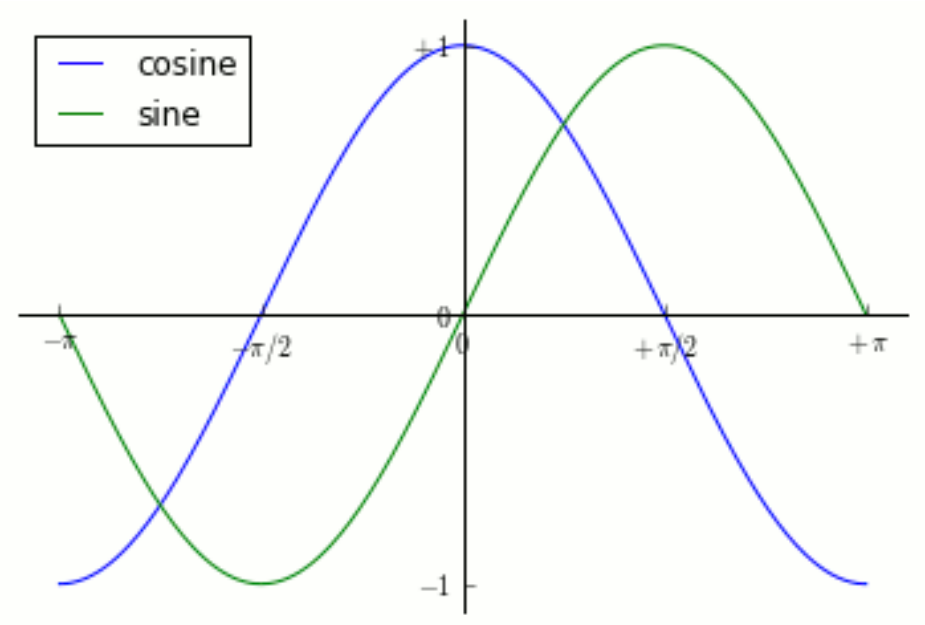
```
>>> plt.plot(x,y,'b--',x,z,'g:')
```



Атрибуты графика (диапазон, деления, легенда)

```
X = np.linspace(-np.pi, np.pi, 256, endpoint=True)
C, S = np.cos(X), np.sin(X)
plt.plot(X, C, color="blue", linestyle="-", label="cosine")
plt.plot(X, S, color="green", linestyle="-", label="sine")
plt.xlim(X.min() * 1.1, X.max() * 1.1)
plt.ylim(C.min() * 1.1, C.max() * 1.1)
plt.xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi],
 [r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'])
plt.yticks([-1, 0, +1], [r'$-1$', r'$0$', r'$+1$'])
ax = plt.gca() # gca -> 'get current axis'
ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['left'].set_position('zero')
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
plt.legend(loc='upper left'); plt.show()
```

Атрибуты графика (пример 2)



Объекты Matplotlib – рисунки, подграфики, оси

Под «рисунком» (figure) в Matplotlib подразумевается целое окно пользовательского интерфейса. Внутри этого рисунка может быть несколько «подграфиков» (subplots). Поле с осями в Matplotlib представлено объектом «оси» (axes). Эти объекты могут создаваться явно или неявно.
Когда вызывается метод plot(), библиотека неявно использует gca() (get current axis), чтобы получить текущее поле с осями, а то, в свою очередь, использует gcf() (get current figure), чтобы получить текущее окно рисунка. Если окно не создано, автоматически вызывается метод figure() для создания окна с параметрами по умолчанию. На созданном окне будет один подграфик – subplot(111).

Объекты Matplotlib – рисунки

Окна-рисунки (figure) нумеруются как в MATLAB от 1.

У окна-рисунка можно переопределить следующие параметры:

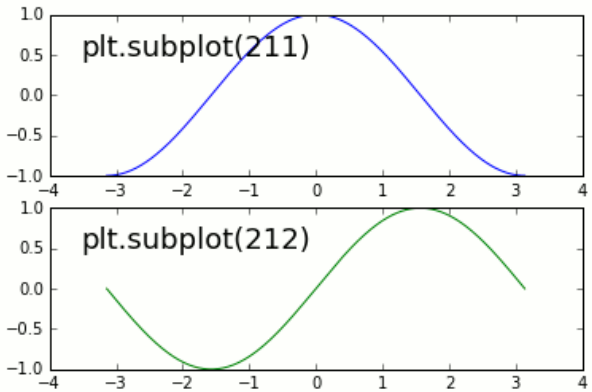
Параметр	По умолчанию	Описание
num	1	Номер окна-рисунка
figsize	figure.figsize	Размер рисунка в дюймах (кортеж)
dpi	figure.dpi	Разрешение рисунка в dpi
facecolor	figure.facecolor	Цвет основного фона
edgecolor	figure.edgecolor	Цвет фона на краях
frameon	True	Рисовать рамку или нет

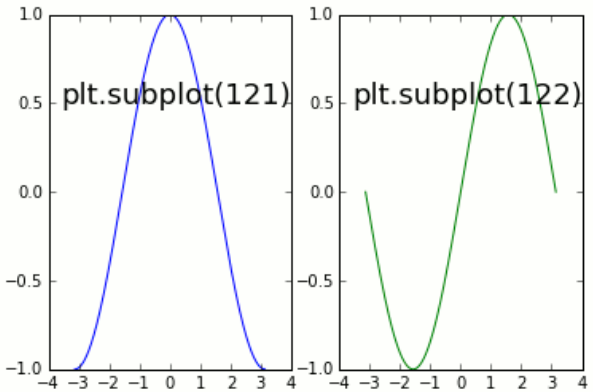
Например:

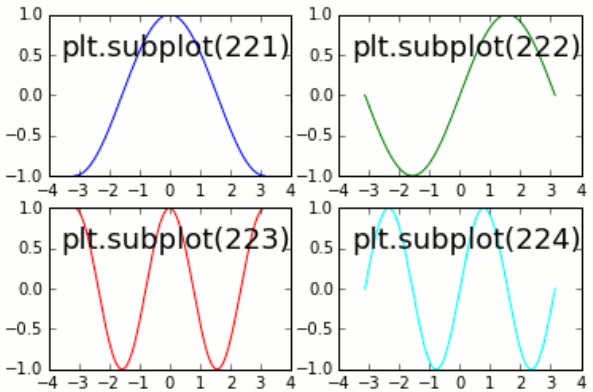
```
plt.figure(figsize=(10, 6), dpi=80)
```

Объекты Matplotlib – подграфики

Используя подграфики, в одном окне можно построить несколько графиков. Подграфики располагаются в виде сетки $N \times M$. При разбиении окна-рисунка на подграфики функцией subplot() у неё указывается три параметра: N (число строк), M (число колонок) и P (номер текущей ячейки). Если все три числа меньше 10, параметром может быть одно трёхзначное число, в котором сотни – N, десятки – M, единицы – P.

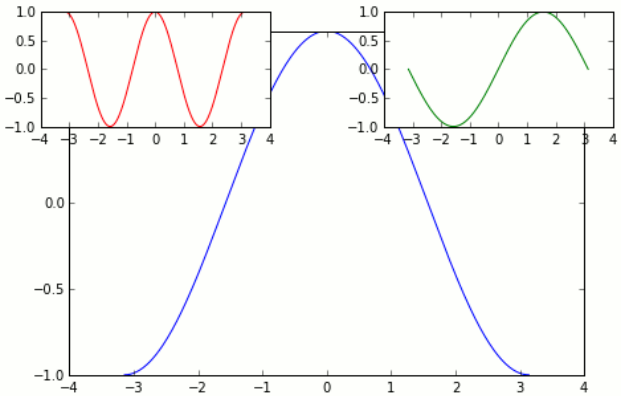






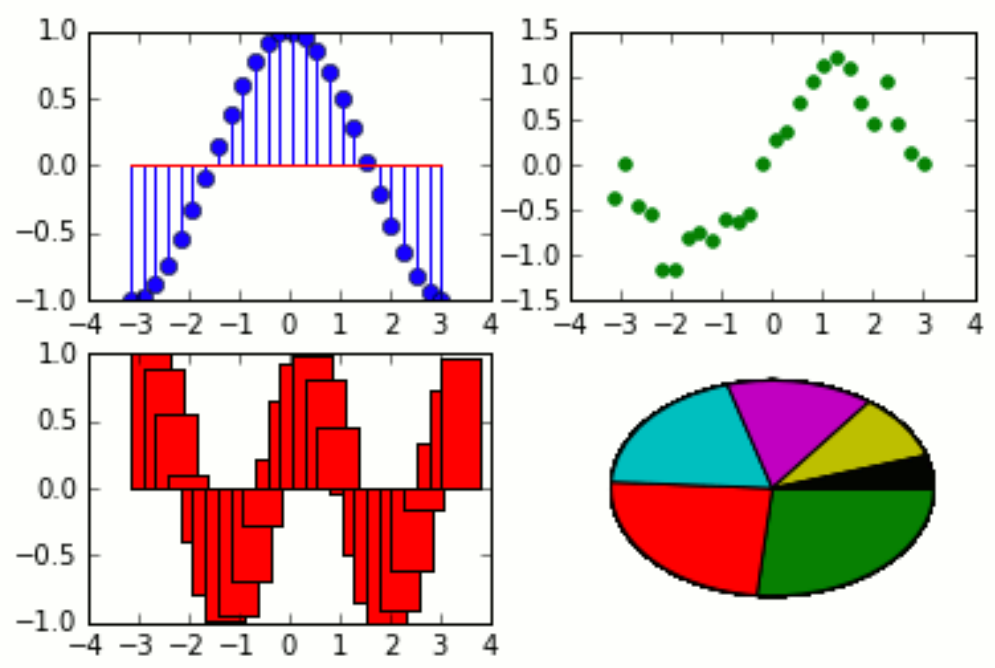
Объекты Matplotlib – оси

Объект «оси» (axes) описывает поле с осями, который так же, как и подграфики, является ещё одним способом разместить несколько графиков в одном окне-рисунке.



```
plt.axes([0.05,0.05,0.9,0.9]) # [left,bottom,width,height]
plt.axes([0,0.7,0.4,0.3])
plt.axes([0.6,0.7,0.4,0.3])
```

Графики разных видов



stem(), scatter(), bar(), pie(), ...

Библиотека SciPy

Линейная алгебра (1)

```
>>> import numpy as np
>>> from scipy import linalg
>>> A = np.array([[1,2],[3,4]])
array([[1, 2],
       [3, 4]])
```

Вычисление определителя матрицы:

```
>>> linalg.det(A)
-2.0
```

Инвертирование матрицы:

```
>>> linalg.inv(A)
array([[-2. ,  1. ],
       [ 1.5, -0.5]])
```

Линейная алгебра (2)

```
>>> import numpy as np
>>> from scipy import linalg
```

Решение СЛАУ:

$$\begin{cases} x_1 + 3x_2 + 5x_3 = 10 \\ 2x_1 + 5x_2 + x_3 = 8 \\ 2x_1 + 3x_2 + 8x_3 = 3 \end{cases}$$

$$\mathrm{A} \cdot \mathrm{x} = \mathrm{b}$$

```
>>> A = np.array([[1,3,5],[2,5,1],[2,3,8]])
>>> b = np.array([10,8,3])
>>> linalg.solve(A,b)
array([-9.28,  5.16,  0.76])
```

Вычисление собственных векторов матриц и др...

Быстрое преобразование Фурье

```
>>> import numpy as np
>>> from scipy.fftpack import fft,ifft
>>> x = np.array([1.0, 2.0, 1.0, -1.0, 1.5])
>>> y = fft(x)
>>> y
array([ 4.50000000+0.j        ,  2.08155948-1.65109876j,
       -1.83155948+1.60822041j, -1.83155948-1.60822041j,
        2.08155948+1.65109876j])
>>> yinv = ifft(y)
>>> yinv
[ 1.0+0.j  2.0+0.j  1.0+0.j -1.0+0.j  1.5+0.j]
```

Статистика и случайные числа

Модуль scipy.stats содержит инструменты статистики и теории вероятностей. (Генераторы псевдослучайных чисел для различных случайных процессов реализованы в модуле numpy.random)

- вычисление моментов случайной величины (мат.ожидания, дисперсии,...);
- вычисление функции распределения (CDF);
- вычисление плотности вероятности (PDF);
- вычисление квантилей и др.

Обработка сигналов

Модуль scipy.signal содержит методы обработки сигналов:

- аппроксимация B-сплайнами;
- вычисление сигнала на выходе линейной системы;
- моделирование цифровых и аналоговых фильтров;
- вычисление периодограмм;
- спектральный анализ и др.