Python for Magnetism Documentation

Release 2020.11

Michael Wack

Nov 03, 2020

CONTENTS

CHAPTER

ONE

CONTENTS:

1.1 Introduction to Python

Python is a versatile, interpreted programming language introduced in 1991. It is available in many varieties and scales very well from little scripts and interactive sessions to full-blown applications with graphical user interfaces. This article on Wikipedia covers some general aspects of Python.

Due to its scalability, easy syntax and free availability it became very popular in science. Therefore many add-on modules for specific problems are available. Those cover many aspects that are traditionally done with other (commercial) tools like numerical calculations, symbolic math and generation of publication ready plots.

1.2 Set up your python environment

1.2.1 Recommended setup

To get Python running on your systems I recommend to install the following packages on your portable computer (by the way: this works also on the stationary computers in our institute without root access):

1. The Python platform Anaconda (make sure to select the version with Python 3.x, *not* 2.x). Available for MacOS, Linux and Windows.

   https://www.continuum.io/downloads

2. The PyCharm IDE: choose the free community edition

   https://www.jetbrains.com/pycharm/download/

3. Get your eduroam WLAN working so that you will have internet access during class. On top you will have free internet access in most universities on the planet . . . .

   https://www.lrz.de/services/netz/mobil/eduroam/

1.2.2 Other useful ressources

If you like a matlab style user interface with interactive variable editor, you can try Spyder IDE: https://pythonhosted.org/spyder/

Official Python web page: https://www.python.org/

Official documentation: https://docs.python.org/3/

1.3 Getting Started

There are many different ways to use Python. For now we will use an iPython notebook.

Create a directory of your choice (where you want to locate your python files). Open a command shell in this directory and start the iPython notebook server by typing `ipython notebook'. This should open your web browser or at least show you the URL that can be opened by the web browser of your choice. Usually http://localhost:8888

In your browser window you can now open or create notebook files (.ipynb) in your directory. Once you opened a notebook you can type 'h' to get help on some useful shortcuts. This text is a static version of such a notebook. You can download it (see start page of this documentation) or type it on your own.

1.3.1 Using Python as a Calculator

```
2+2
```

```
4
```

```
50-5*6
```

```
20
```

```
(50-5*6) / 4
```

```
5.0
```

```
8/5
```

```
1.6
```

```
8//5
```

```
1
```

```
8%5
```

```
3
```

1.3.2 Variables and Types

Naming

Do not use reserved names:

and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield

Use a-z, A-Z, 0-9, _

Start with a letter

```
myvar = 12
```

```
myvar
```

```
12
```

```
myvar + 10
```

```
22
```

```
myvar = myvar + 10
```

```
myvar
```

```
22
```

```
type(myvar)
```

```
int
```

```
myvar = 22.0
```

```
type(myvar)
```

```
float
```

```
mybool = True
```

```
type( mybool)
```

```
bool
```

```
x = 1.0 + 1.0j
```

```
type( x)
```

```
complex
```

```
x.real
```

```
1.0
```

```
myvar**0.5
```

```
4.69041575982343
```

type casting

```
type(myvar)
```

```
float
```

```
myvar=22.9
```

```
int(myvar)
```

```
22
```

Comparision

```
1 > 0
```

```
True
```

```
type(0 > 1)
```

```
bool
```

```
2 >= 2
```

```
True
```

```
not myvar != 1
```

```
False
```

```
(1==1) | (1==2)
```

```
True
```

Composite types: Strings, List, Tuple, Set and Dictionaries

Strings

```
s = "Hello Everybody"
```

```
type(s)
```

```
str
```

```
s
```

```
'Hello Everybody'
```

```
len(s)
```

```
15
```

```
s.replace('Hello', 'Goodbye')
```

```
'Goodbye Everybody'
```

```
s[0]
```

```
'H'
```

```
s[6:]
```

```
'Everybody'
```

```
s[0:1]
```

```
'H'
```

```
s[:-3]
```

```
'Hello Everyb'
```

```
s[0:10:2]
```

```
'HloEe'
```

```
s[::-2]
```

```
'yoyeEolH'
```

```
s2 = ' and Nobody'
```

```
s + s2
```

```
'Hello Everybody and Nobody'
```

```
s[0:5] + s2[-7:]
```

```
'Hello Nobody'
```

```
print( "The number {} is great".format(myvar))
```

```
The number 22.9 is great
```

Lists

```
l = [1,2,3,4]
```

```
l
```

```
[1, 2, 3, 4]
```

```
type(l)
```

```
list
```

```
l2 = list(range(10))
```

```
l2
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
l5 = list(range(-10,11,2))
```

```
l5
```

```
[-10, -8, -6, -4, -2, 0, 2, 4, 6, 8, 10]
```

```
l3 = [1.1, 2, 'Hallo', 1+1j]
```

```
l3
```

```
[1.1, 2, 'Hallo', (1+1j)]
```

```
l3[1]
```

```
2
```

```
l4 = [1,2,3,[1,2,3,[5,6,7]]]
```

```
l4
```

```
[1, 2, 3, [1, 2, 3, [5, 6, 7]]]
```

```
l4[3][3]
```

```
[5, 6, 7]
```

```
l4[3][3][2]
```

```
7
```

```
ls = list(s)
```

```
ls
```

```
['H', 'e', 'l', 'l', 'o', ' ', 'E', 'v', 'e', 'r', 'y', 'b', 'o', 'd', 'y']
```

```
ls.sort()
```

```
ls
```

```
[' ', 'E', 'H', 'b', 'd', 'e', 'e', 'l', 'l', 'o', 'o', 'r', 'v', 'y', 'y']
```

```
ls.append( 'X')
```

```
l6 = []
```

```
l6
```

```
[]
```

```
l6.append( 'H')
```

```
l6.append('a')
```

```
l6
```

```
['H', 'a']
```

```
l6.insert( 0, 'X')
```

```
l6
```

```
['X', 'H', 'a']
```

```
l6.insert( 2, 'A')
```

```
l6
```

```
['X', 'H', 'A', 'a']
```

```
l6[0] = 'x'
```

```
l6
```

```
['x', 'H', 'A', 'a']
```

Tuple

similiar to list but it is impossible to modify

```
t1 = (1,2,3,4)
```

```
t1
```

```
(1, 2, 3, 4)
```

```
# t1[0] = 3
```

Dictionaries

```
d1 = {'par1': 5, 'par2': 10}
```

```
d1
```

```
{'par1': 5, 'par2': 10}
```

```
type(d1)
```

```
dict
```

```
d1['par1']
```

```
5
```

```
d1['par1']=1
```

```
d1['par3']=22
```

```
d1
```

```
{'par1': 1, 'par2': 10, 'par3': 22}
```

Set

```
basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
```

```
print(basket)
```

```
{'banana', 'apple', 'pear', 'orange'}
```

```
'orange' in basket
```

```
True
```

```
'lemon' in basket
```

```
False
```

```
a = set('abracadabra')
b = set('alacazam')
```

```
a, b
```

```
({'a', 'b', 'c', 'd', 'r'}, {'a', 'c', 'l', 'm', 'z'})
```

```
a-b
```

```
{'b', 'd', 'r'}
```

```
a & b
```

```
{'a', 'c'}
```

```
a | b
```

```
{'a', 'b', 'c', 'd', 'l', 'm', 'r', 'z'}
```

1.4 Program Flow

1.4.1 if, else, elif

```
cond1, cond2 = True, False
```

```
if cond1:
    print("cond1 is True")

elif cond2:
    print("will never happen if dond1 is True")

else:
    print("cond1 and cond2 are False")
```

```
cond1 is True
```

Please note that program blocks are defined by their indentation level.

```
if cond1:
    if not cond2:
        print("cond1 is True and cond2 is False")

print("I'm always here!")
```

```
cond1 is True and cond2 is False
I'm always here!
```

```
if cond1 and not cond2:
    print("cond1 is True and cond2 is False")
```

```
cond1 is True and cond2 is False
```

1.4.2 Loops

for

for loops execute an program block for each element in a list:

```
for c in range(5):
    print(c+1)
```

```
1
2
3
4
5
```

```
for letter in 'Hallo':
    print('The letter is: {}'.format(letter))
```

```
The letter is: H
The letter is: a
The letter is: l
The letter is: l
The letter is: o
```

```
for magnetic_mineral in ('magnetite', 'hematite', 'greigite'):
    print( magnetic_mineral)
```

```
magnetite
hematite
greigite
```

enumerate give access to the indices when needed

```
for idx, c in enumerate(range(-3,3)):
    print(idx, c, idx*c)
```

```
0 -3 0
1 -2 -2
2 -1 -2
3 0 0
4 1 4
5 2 10
```

```
for x in 'ABCDE':
    for y in range(4):
        print( "{}{}".format(x,y))
```

```
A0
A1
A2
A3
B0
B1
B2
B3
C0
C1
C2
C3
D0
D1
D2
D3
E0
E1
E2
E3
```

List comprehensions

instead of doing something like this:

```
cubes = []
for c in range(4):
    cubes.append(c**3)
print(cubes)
```

```
[0, 1, 8, 27]
```

we can write:

```
[n**3 for n in range(4)]
```

```
[0, 1, 8, 27]
```

```
[(x,x**2) for x in range(1,5)]
```

```
[(1, 1), (2, 4), (3, 9), (4, 16)]
```

```
[s.upper() for s in ['small', 'little', 'tiny']]
```

```
['SMALL', 'LITTLE', 'TINY']
```

```
["{}{}".format(x,y) for x in "ABCDE" for y in range(4)]
```

```
['A0',
 'A1',
 'A2',
 'A3',
 'B0',
 'B1',
 'B2',
 'B3',
 'C0',
 'C1',
 'C2',
 'C3',
 'D0',
 'D1',
 'D2',
 'D3',
 'E0',
 'E1',
 'E2',
 'E3']
```

```
[x for x in range(10) if x > 3 and x < 8]
```

```
[4, 5, 6, 7]
```

```
vec = [[1,2,3], [4,5,6], [7,8,9]]
[num for elem in vec for num in elem if num % 2 == 0]
```

```
[2, 4, 6, 8]
```

Dictionary comprehensions

```
{x.upper(): x*3 for x in 'abcd'}
```

```
{'A': 'aaa', 'B': 'bbb', 'C': 'ccc', 'D': 'ddd'}
```

while

```
c = 0
while c < 5:
    c += 1
    print(c)
print("done")
```

```
1
2
3
4
5
done
```

1.4.3 Functions

```
def f():
    print("got called")
```

```
f()
```

```
got called
```

```
def sqrt(n):
    """
    return square root of n
    """
    return n**0.5
```

```
sqrt(2)
```

```
1.4142135623730951
```

```
def mul(a,b):
    """
    multiply a and b
    """
    return a*b

mul(2,3)
```

```
6
```

```
def div(a,b):
    """
    divide a by b
    return int division and reminder
    """
    return a // b, a % b

div(20,6)
```

```
(3, 2)
```

docstrings can be retrieved by builtin help function

```
help(div)
```

```
Help on function div in module __main__:

div(a, b)
    divide a by b
    return int division and reminder
```

works with any object that has a doc string

```
help(sorted)
```

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

sorted(...)
    sorted(iterable, key=None, reverse=False) --> new sorted list
```

```
def add(a,b=1):
    """
    add a and b
    if b is not given it is assumed to be 1
    """
    return a+b

print(add(2,3))
print(add(7))
```

```
5
8
```

anonymous functions: lambda

```
square = lambda n: n**2
```

```
square(3)
```

```
9
```

for efficient computation, functions can be applied to a list of parameters

```
map( square, range(10))
```

```
<map at 0x7f1394041c18>
```

```
map( lambda x: x**3, range(10))
```

```
<map at 0x7f1394041e10>
```

1.4.4 Exceptions

exceptions output error messages

```
1+a
```

```
---------------------------------------------------------------------------

NameError                                 Traceback (most recent call last)

<ipython-input-33-f9dd4f57d0fd> in <module>()
----> 1 1+a


NameError: name 'a' is not defined
```

```
a = 2
a[1]
```

```
---------------------------------------------------------------------------

TypeError                                 Traceback (most recent call last)

<ipython-input-34-79d66ab95ee2> in <module>()
      1 a = 2
----> 2 a[1]


TypeError: 'int' object is not subscriptable
```

```
try:
    zz *= 2
except NameError:
    print( "Caught NameError: zz not known")
```

```
Caught NameError: zz not known
```

1.4.5 Modules and Namespaces

```
import math
```

```
math.cos( math.radians(45))
```

```
0.7071067811865476
```

```
from math import sin, degrees
```

```
sin( degrees(45))
```

```
0.8060754911159176
```

```
dir(math)
```

```
['__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'acos',
 'acosh',
 'asin',
 'asinh',
 'atan',
 'atan2',
 'atanh',
 'ceil',
```

(continues on next page)

(continued from previous page)

```
 'copysign',
 'cos',
 'cosh',
 'degrees',
 'e',
 'erf',
 'erfc',
 'exp',
 'expm1',
 'fabs',
 'factorial',
 'floor',
 'fmod',
 'frexp',
 'fsum',
 'gamma',
 'hypot',
 'isfinite',
 'isinf',
 'isnan',
 'ldexp',
 'lgamma',
 'log',
 'log10',
 'log1p',
 'log2',
 'modf',
 'pi',
 'pow',
 'radians',
 'sin',
 'sinh',
 'sqrt',
 'tan',
 'tanh',
 'trunc']
```

```
help(math)
```

```
Help on module math:

NAME
    math

MODULE REFERENCE
    http://docs.python.org/3.4/library/math

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    This module is always available.  It provides access to the
    mathematical functions defined by the C standard.
```

(continues on next page)

(continued from previous page)

```
FUNCTIONS
    acos(...)
        acos(x)

        Return the arc cosine (measured in radians) of x.

    acosh(...)
        acosh(x)

        Return the inverse hyperbolic cosine of x.

    asin(...)
        asin(x)

        Return the arc sine (measured in radians) of x.

    asinh(...)
        asinh(x)

        Return the inverse hyperbolic sine of x.

    atan(...)
        atan(x)

        Return the arc tangent (measured in radians) of x.

    atan2(...)
        atan2(y, x)

        Return the arc tangent (measured in radians) of y/x.
        Unlike atan(y/x), the signs of both x and y are considered.

    atanh(...)
        atanh(x)

        Return the inverse hyperbolic tangent of x.

    ceil(...)
        ceil(x)

        Return the ceiling of x as an int.
        This is the smallest integral value >= x.

    copysign(...)
        copysign(x, y)

        Return a float with the magnitude (absolute value) of x but the sign
        of y. On platforms that support signed zeros, copysign(1.0, -0.0)
        returns -1.0.

    cos(...)
        cos(x)

        Return the cosine of x (measured in radians).

    cosh(...)
```

(continues on next page)

(continued from previous page)

```
        cosh(x)

        Return the hyperbolic cosine of x.

    degrees(...)
        degrees(x)

        Convert angle x from radians to degrees.

    erf(...)
        erf(x)

        Error function at x.

    erfc(...)
        erfc(x)

        Complementary error function at x.

    exp(...)
        exp(x)

        Return e raised to the power of x.

    expm1(...)
        expm1(x)

        Return exp(x)-1.
        This function avoids the loss of precision involved in the direct evaluation␣
↪of exp(x)-1 for small x.

    fabs(...)
        fabs(x)

        Return the absolute value of the float x.

    factorial(...)
        factorial(x) -> Integral

        Find x!. Raise a ValueError if x is negative or non-integral.

    floor(...)
        floor(x)

        Return the floor of x as an int.
        This is the largest integral value <= x.

    fmod(...)
        fmod(x, y)

        Return fmod(x, y), according to platform C.  x % y may differ.

    frexp(...)
        frexp(x)

        Return the mantissa and exponent of x, as pair (m, e).
        m is a float and e is an int, such that x = m * 2.**e.
```

(continues on next page)

(continued from previous page)

```
        If x is 0, m and e are both 0.  Else 0.5 <= abs(m) < 1.0.

    fsum(...)
        fsum(iterable)

        Return an accurate floating point sum of values in the iterable.
        Assumes IEEE-754 floating point arithmetic.

    gamma(...)
        gamma(x)

        Gamma function at x.

    hypot(...)
        hypot(x, y)

        Return the Euclidean distance, sqrt(x*x + y*y).

    isfinite(...)
        isfinite(x) -> bool

        Return True if x is neither an infinity nor a NaN, and False otherwise.

    isinf(...)
        isinf(x) -> bool

        Return True if x is a positive or negative infinity, and False otherwise.

    isnan(...)
        isnan(x) -> bool

        Return True if x is a NaN (not a number), and False otherwise.

    ldexp(...)
        ldexp(x, i)

        Return x * (2**i).

    lgamma(...)
        lgamma(x)

        Natural logarithm of absolute value of Gamma function at x.

    log(...)
        log(x[, base])

        Return the logarithm of x to the given base.
        If the base not specified, returns the natural logarithm (base e) of x.

    log10(...)
        log10(x)

        Return the base 10 logarithm of x.

    log1p(...)
        log1p(x)
```

(continues on next page)

(continued from previous page)

```
        Return the natural logarithm of 1+x (base e).
        The result is computed in a way which is accurate for x near zero.

    log2(...)
        log2(x)

        Return the base 2 logarithm of x.

    modf(...)
        modf(x)

        Return the fractional and integer parts of x.  Both results carry the sign
        of x and are floats.

    pow(...)
        pow(x, y)

        Return x**y (x to the power of y).

    radians(...)
        radians(x)

        Convert angle x from degrees to radians.

    sin(...)
        sin(x)

        Return the sine of x (measured in radians).

    sinh(...)
        sinh(x)

        Return the hyperbolic sine of x.

    sqrt(...)
        sqrt(x)

        Return the square root of x.

    tan(...)
        tan(x)

        Return the tangent of x (measured in radians).

    tanh(...)
        tanh(x)

        Return the hyperbolic tangent of x.

    trunc(...)
        trunc(x:Real) -> Integral

        Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic method.

DATA
    e = 2.718281828459045
    pi = 3.141592653589793
```

(continues on next page)

(continued from previous page)

```
FILE
    /home/wack/bin/anaconda3/lib/python3.4/lib-dynload/math.cpython-34m.so
```

```
help(math.sqrt)
```

```
Help on built-in function sqrt in module math:

sqrt(...)
    sqrt(x)

    Return the square root of x.
```

1.5 Pylab - Matlab style Python

Pylab is a module that provides a Matlab like namespace by importing functions from the modules Numpy and Matplotlib. Numpy provides efficient numerical vector calculations based on underlying Fortran and C binary libraries. Matplotlib contains functions to create visualizations of data.

```
# this ipython magic command imports pylab and allows the plots to reside within the␣
↪notebook
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
Value error parsing header in AFM: b'UnderlinePosition' b'-133rUnderlineThickness␣
↪20rVersion 7.000'
```

The following examples are based on Florian Lhuillier's lecture on Matlab which can be found online:

http://geophysik.uni-muenchen.de/~lhuillier/teaching/AD2010/

1.5.1 Examples based on "Rappels d'utilisation de MATLAB"

numpy.array

```
M = array(((1,2,3), (4,5,6), (7,8,9)))
```

```
M
```

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

```
L=array((1,2,3))
```

```
C=array((1,2,3)).T
```

```
L, C
```

```
(array([1, 2, 3]), array([1, 2, 3]))
```

```
t1=arange(1,11); t1
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
t2=arange(1,11,2); t2
```

```
array([1, 3, 5, 7, 9])
```

```
t3=arange(10,0,-1); t3
```

```
array([10,  9,  8,  7,  6,  5,  4,  3,  2,  1])
```

get specific elements

```
t1
```

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
t1[4]
```

```
5
```

```
t1[[0,4,9]]
```

```
array([ 1,  5, 10])
```

```
t1[0:5]
```

```
array([1, 2, 3, 4, 5])
```

```
t1[0::2]
```

```
array([1, 3, 5, 7, 9])
```

```
t1[1::2]
```

```
array([ 2,  4,  6,  8, 10])
```

```
t1>5
```

```
array([False, False, False, False, False,  True,  True,  True,  True,  True],␣
↪dtype=bool)
```

```
t1[t1>5]
```

```
array([ 6,  7,  8,  9, 10])
```

```
t1%2==0
```

```
array([False,  True, False,  True, False,  True, False,  True, False,  True],␣
↪dtype=bool)
```

```
t1[t1%2==0]
```

```
array([ 2,  4,  6,  8, 10])
```

```
t1%2==1
```

```
array([ True, False,  True, False,  True, False,  True, False,  True, False],␣
↪dtype=bool)
```

```
t1[t1%2==1]
```

```
array([1, 3, 5, 7, 9])
```

Matplotlib plotting

```
t = arange( -2*pi,2*pi,0.1); t
```

```
array([-6.28318531, -6.18318531, -6.08318531, -5.98318531, -5.88318531,
       -5.78318531, -5.68318531, -5.58318531, -5.48318531, -5.38318531,
       -5.28318531, -5.18318531, -5.08318531, -4.98318531, -4.88318531,
       -4.78318531, -4.68318531, -4.58318531, -4.48318531, -4.38318531,
       -4.28318531, -4.18318531, -4.08318531, -3.98318531, -3.88318531,
       -3.78318531, -3.68318531, -3.58318531, -3.48318531, -3.38318531,
       -3.28318531, -3.18318531, -3.08318531, -2.98318531, -2.88318531,
       -2.78318531, -2.68318531, -2.58318531, -2.48318531, -2.38318531,
       -2.28318531, -2.18318531, -2.08318531, -1.98318531, -1.88318531,
       -1.78318531, -1.68318531, -1.58318531, -1.48318531, -1.38318531,
       -1.28318531, -1.18318531, -1.08318531, -0.98318531, -0.88318531,
       -0.78318531, -0.68318531, -0.58318531, -0.48318531, -0.38318531,
       -0.28318531, -0.18318531, -0.08318531,  0.01681469,  0.11681469,
        0.21681469,  0.31681469,  0.41681469,  0.51681469,  0.61681469,
        0.71681469,  0.81681469,  0.91681469,  1.01681469,  1.11681469,
        1.21681469,  1.31681469,  1.41681469,  1.51681469,  1.61681469,
        1.71681469,  1.81681469,  1.91681469,  2.01681469,  2.11681469,
        2.21681469,  2.31681469,  2.41681469,  2.51681469,  2.61681469,
        2.71681469,  2.81681469,  2.91681469,  3.01681469,  3.11681469,
        3.21681469,  3.31681469,  3.41681469,  3.51681469,  3.61681469,
        3.71681469,  3.81681469,  3.91681469,  4.01681469,  4.11681469,
        4.21681469,  4.31681469,  4.41681469,  4.51681469,  4.61681469,
        4.71681469,  4.81681469,  4.91681469,  5.01681469,  5.11681469,
        5.21681469,  5.31681469,  5.41681469,  5.51681469,  5.61681469,
        5.71681469,  5.81681469,  5.91681469,  6.01681469,  6.11681469,
        6.21681469])
```

```
x1=cos(t)
```

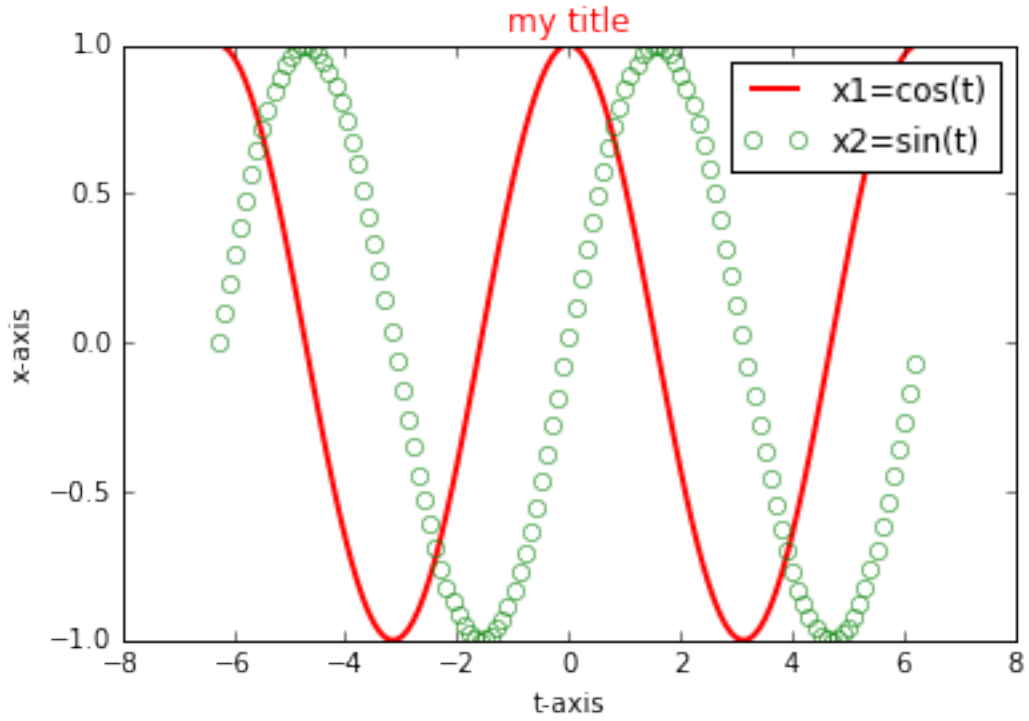
```
x2=sin(t)
```

```
len(x2)
```

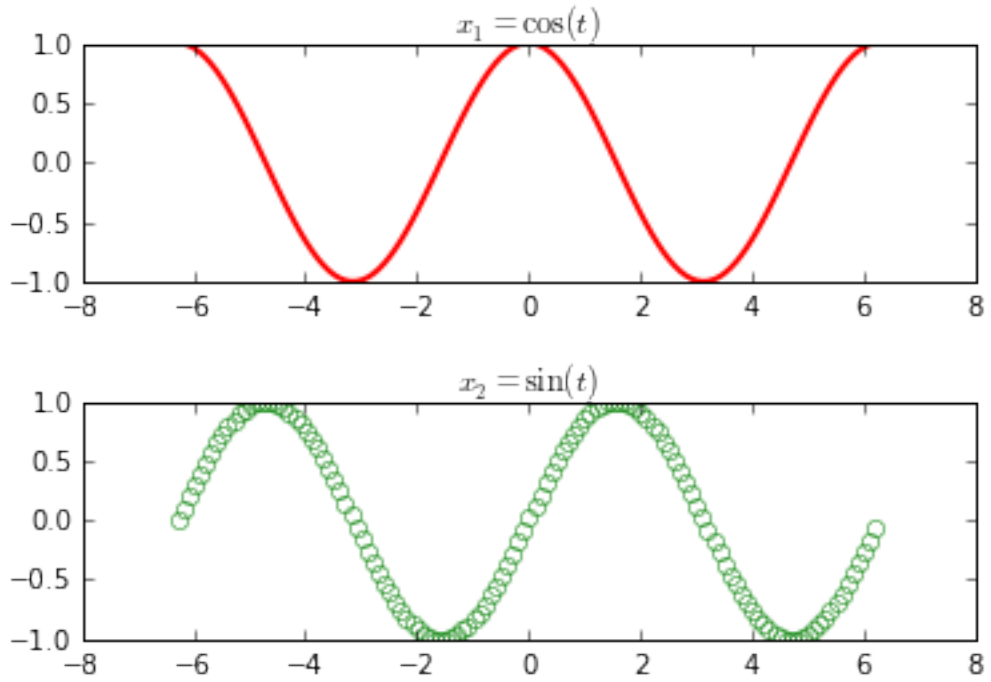
```
126
```

```
plot(t,x1, color='r', linewidth=2, label='x1=cos(t)')
plot(t, x2, ' o', markeredgecolor='g', markerfacecolor="None", label='x2=sin(t)')
title('my title', color='r')
xlabel('t-axis')
ylabel('x-axis')
legend()
```

```
<matplotlib.legend.Legend at 0x7f5fe9c0dc18>
```




```
subplot(211)
plot(t, x1, 'r', linewidth=2)
title('$x_1=\cos(t)$')
subplot(212)
plot(t, x2, ' o', markeredgecolor='g', markerfacecolor="None")
title('$x_2=\sin(t)$')
subplots_adjust(hspace=.5)
```



1.5.2 Examples based on "familiarisation avec Matlab"

```
set_printoptions(precision=2, suppress=True)
```

Excercise 1

```
x=array((17,8,12,15,6,11,9,18,16,10,13,19)); x
```

```
array([17,  8, 12, 15,  6, 11,  9, 18, 16, 10, 13, 19])
```

```
N = len(x); N
```

```
12
```

```
S = sum(x); S
```

```
154
```

```
xbarre = S / N; xbarre
```

```
12.833333333333334
```

```
xbarre = mean( x); xbarre
```

```
12.833333333333334
```

```
sigma = sqrt(sum((x-xbarre)**2)/(N-1)); sigma
```

```
4.1959576506514704
```

```
sigma = std(x, ddof=1); sigma
```

```
4.1959576506514704
```

```
dx = x[1:]-x[0:-1]; dx
```

```
array([-9,  4,  3, -9,  5, -2,  9, -2, -6,  3,  6])
```

```
dx = diff(x); dx
```

```
array([-9,  4,  3, -9,  5, -2,  9, -2, -6,  3,  6])
```

Excercise 2

```
t = linspace(-25,25,51); t
```

```
array([-25., -24., -23., -22., -21., -20., -19., -18., -17., -16., -15.,
       -14., -13., -12., -11., -10.,  -9.,  -8.,  -7.,  -6.,  -5.,  -4.,
        -3.,  -2.,  -1.,   0.,   1.,   2.,   3.,   4.,   5.,   6.,   7.,
         8.,   9.,  10.,  11.,  12.,  13.,  14.,  15.,  16.,  17.,  18.,
        19.,  20.,  21.,  22.,  23.,  24.,  25.])
```

```
x = t**2; x
```

```
array([ 625.,  576.,  529.,  484.,  441.,  400.,  361.,  324.,  289.,
        256.,  225.,  196.,  169.,  144.,  121.,  100.,   81.,   64.,
         49.,   36.,   25.,   16.,    9.,    4.,    1.,    0.,    1.,
          4.,    9.,   16.,   25.,   36.,   49.,   64.,   81.,  100.,
        121.,  144.,  169.,  196.,  225.,  256.,  289.,  324.,  361.,
        400.,  441.,  484.,  529.,  576.,  625.])
```

```
y = t[::-1]**3; y
```

```
array([ 15625.,  13824.,  12167.,  10648.,   9261.,   8000.,   6859.,
         5832.,   4913.,   4096.,   3375.,   2744.,   2197.,   1728.,
         1331.,   1000.,    729.,    512.,    343.,    216.,    125.,
           64.,     27.,      8.,      1.,      0.,     -1.,     -8.,
          -27.,    -64.,   -125.,   -216.,   -343.,   -512.,   -729.,
        -1000.,  -1331.,  -1728.,  -2197.,  -2744.,  -3375.,  -4096.,
        -4913.,  -5832.,  -6859.,  -8000.,  -9261., -10648., -12167.,
       -13824., -15625.])
```

```
fliplr(t[newaxis])**3
```

```
array([[ 15625.,  13824.,  12167.,  10648.,   9261.,   8000.,   6859.,
          5832.,   4913.,   4096.,   3375.,   2744.,   2197.,   1728.,
```

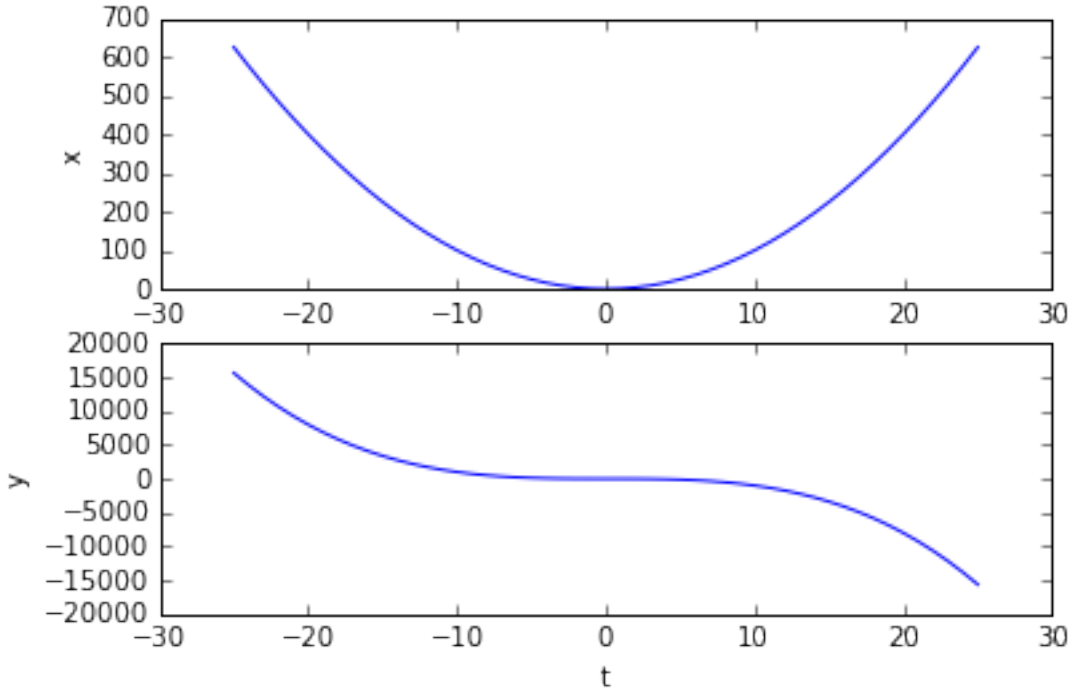
(continues on next page)

(continued from previous page)

```
          1331.,   1000.,    729.,    512.,    343.,    216.,    125.,
            64.,     27.,      8.,      1.,      0.,     -1.,     -8.,
           -27.,    -64.,   -125.,   -216.,   -343.,   -512.,   -729.,
         -1000.,  -1331.,  -1728.,  -2197.,  -2744.,  -3375.,  -4096.,
         -4913.,  -5832.,  -6859.,  -8000.,  -9261., -10648., -12167.,
        -13824., -15625.]])
```

```
subplot(2,1,1)
plot(t,x)
xlabel('t'); ylabel('x')
subplot(212)
plot(t,y)
xlabel('t'); ylabel('y')
```

```
<matplotlib.text.Text at 0x7f5fe95ac978>
```




```
fix(x/2)==x/2
```

```
array([False,  True, False,  True, False,  True, False,  True, False,
        True, False,  True, False,  True, False,  True, False,  True,
       False,  True, False,  True, False,  True, False,  True, False,
        True, False,  True, False,  True, False,  True, False,  True,
       False,  True, False,  True, False,  True, False,  True, False,
        True, False,  True, False,  True, False], dtype=bool)
```

```
sum( x[fix(x/2)==x/2])
```

```
5200.0
```

```
sum( x[remainder(x,2)==0])
```

```
5200.0
```

```
sum( x[x%2==0])
```

```
5200.0
```

```
sum( y[y>0])
```

```
105625.0
```

Excercise 3

```
t = arange(1,10.01,0.5); t
```

```
array([  1. ,   1.5,   2. ,   2.5,   3. ,   3.5,   4. ,   4.5,   5. ,
         5.5,   6. ,   6.5,   7. ,   7.5,   8. ,   8.5,   9. ,   9.5,  10. ])
```

```
A = array( (t, t**2, t**3, t**4)).T; A
```

```
array([[     1.  ,      1.  ,      1.  ,      1.  ],
       [     1.5 ,      2.25,      3.38,      5.06],
       [     2.  ,      4.  ,      8.  ,     16.  ],
       [     2.5 ,      6.25,     15.62,     39.06],
       [     3.  ,      9.  ,     27.  ,     81.  ],
       [     3.5 ,     12.25,     42.88,    150.06],
       [     4.  ,     16.  ,     64.  ,    256.  ],
       [     4.5 ,     20.25,     91.12,    410.06],
       [     5.  ,     25.  ,    125.  ,    625.  ],
       [     5.5 ,     30.25,    166.38,    915.06],
       [     6.  ,     36.  ,    216.  ,   1296.  ],
       [     6.5 ,     42.25,    274.62,   1785.06],
       [     7.  ,     49.  ,    343.  ,   2401.  ],
       [     7.5 ,     56.25,    421.88,   3164.06],
       [     8.  ,     64.  ,    512.  ,   4096.  ],
       [     8.5 ,     72.25,    614.12,   5220.06],
       [     9.  ,     81.  ,    729.  ,   6561.  ],
       [     9.5 ,     90.25,    857.38,   8145.06],
       [    10.  ,    100.  ,   1000.  ,  10000.  ]])
```

```
A = column_stack((A, t>5)); A
```

```
array([[     1.  ,      1.  ,      1.  ,      1.  ,      0.  ],
       [     1.5 ,      2.25,      3.38,      5.06,      0.  ],
       [     2.  ,      4.  ,      8.  ,     16.  ,      0.  ],
       [     2.5 ,      6.25,     15.62,     39.06,      0.  ],
       [     3.  ,      9.  ,     27.  ,     81.  ,      0.  ],
       [     3.5 ,     12.25,     42.88,    150.06,      0.  ],
       [     4.  ,     16.  ,     64.  ,    256.  ,      0.  ],
       [     4.5 ,     20.25,     91.12,    410.06,      0.  ],
```

(continues on next page)

(continued from previous page)

```
       [     5.  ,     25.  ,    125.  ,    625.  ,      0.  ],
       [     5.5 ,     30.25,    166.38,    915.06,      1.  ],
       [     6.  ,     36.  ,    216.  ,   1296.  ,      1.  ],
       [     6.5 ,     42.25,    274.62,   1785.06,      1.  ],
       [     7.  ,     49.  ,    343.  ,   2401.  ,      1.  ],
       [     7.5 ,     56.25,    421.88,   3164.06,      1.  ],
       [     8.  ,     64.  ,    512.  ,   4096.  ,      1.  ],
       [     8.5 ,     72.25,    614.12,   5220.06,      1.  ],
       [     9.  ,     81.  ,    729.  ,   6561.  ,      1.  ],
       [     9.5 ,     90.25,    857.38,   8145.06,      1.  ],
       [    10.  ,    100.  ,   1000.  ,  10000.  ,      1.  ]])
```

```
column_stack((A, (fix(t)==t)*5))
```

```
array([[     1.  ,      1.  ,      1.  ,      1.  ,      0.  ,      5.  ],
       [     1.5 ,      2.25,      3.38,      5.06,      0.  ,      0.  ],
       [     2.  ,      4.  ,      8.  ,     16.  ,      0.  ,      5.  ],
       [     2.5 ,      6.25,     15.62,     39.06,      0.  ,      0.  ],
       [     3.  ,      9.  ,     27.  ,     81.  ,      0.  ,      5.  ],
       [     3.5 ,     12.25,     42.88,    150.06,      0.  ,      0.  ],
       [     4.  ,     16.  ,     64.  ,    256.  ,      0.  ,      5.  ],
       [     4.5 ,     20.25,     91.12,    410.06,      0.  ,      0.  ],
       [     5.  ,     25.  ,    125.  ,    625.  ,      0.  ,      5.  ],
       [     5.5 ,     30.25,    166.38,    915.06,      1.  ,      0.  ],
       [     6.  ,     36.  ,    216.  ,   1296.  ,      1.  ,      5.  ],
       [     6.5 ,     42.25,    274.62,   1785.06,      1.  ,      0.  ],
       [     7.  ,     49.  ,    343.  ,   2401.  ,      1.  ,      5.  ],
       [     7.5 ,     56.25,    421.88,   3164.06,      1.  ,      0.  ],
       [     8.  ,     64.  ,    512.  ,   4096.  ,      1.  ,      5.  ],
       [     8.5 ,     72.25,    614.12,   5220.06,      1.  ,      0.  ],
       [     9.  ,     81.  ,    729.  ,   6561.  ,      1.  ,      5.  ],
       [     9.5 ,     90.25,    857.38,   8145.06,      1.  ,      0.  ],
       [    10.  ,    100.  ,   1000.  ,  10000.  ,      1.  ,      5.  ]])
```

```
column_stack((A, (remainder(t,1)==0)*5))
```

```
array([[     1.  ,      1.  ,      1.  ,      1.  ,      0.  ,      5.  ],
       [     1.5 ,      2.25,      3.38,      5.06,      0.  ,      0.  ],
       [     2.  ,      4.  ,      8.  ,     16.  ,      0.  ,      5.  ],
       [     2.5 ,      6.25,     15.62,     39.06,      0.  ,      0.  ],
       [     3.  ,      9.  ,     27.  ,     81.  ,      0.  ,      5.  ],
       [     3.5 ,     12.25,     42.88,    150.06,      0.  ,      0.  ],
       [     4.  ,     16.  ,     64.  ,    256.  ,      0.  ,      5.  ],
       [     4.5 ,     20.25,     91.12,    410.06,      0.  ,      0.  ],
       [     5.  ,     25.  ,    125.  ,    625.  ,      0.  ,      5.  ],
       [     5.5 ,     30.25,    166.38,    915.06,      1.  ,      0.  ],
       [     6.  ,     36.  ,    216.  ,   1296.  ,      1.  ,      5.  ],
       [     6.5 ,     42.25,    274.62,   1785.06,      1.  ,      0.  ],
       [     7.  ,     49.  ,    343.  ,   2401.  ,      1.  ,      5.  ],
       [     7.5 ,     56.25,    421.88,   3164.06,      1.  ,      0.  ],
       [     8.  ,     64.  ,    512.  ,   4096.  ,      1.  ,      5.  ],
       [     8.5 ,     72.25,    614.12,   5220.06,      1.  ,      0.  ],
       [     9.  ,     81.  ,    729.  ,   6561.  ,      1.  ,      5.  ],
       [     9.5 ,     90.25,    857.38,   8145.06,      1.  ,      0.  ],
       [    10.  ,    100.  ,   1000.  ,  10000.  ,      1.  ,      5.  ]])
```

```
column_stack((A, (mod(t,1)==0)*5))
```

```
array([[     1.  ,      1.  ,      1.  ,      1.  ,      0.  ,      5.  ],
       [     1.5 ,      2.25,      3.38,      5.06,      0.  ,      0.  ],
       [     2.  ,      4.  ,      8.  ,     16.  ,      0.  ,      5.  ],
       [     2.5 ,      6.25,     15.62,     39.06,      0.  ,      0.  ],
       [     3.  ,      9.  ,     27.  ,     81.  ,      0.  ,      5.  ],
       [     3.5 ,     12.25,     42.88,    150.06,      0.  ,      0.  ],
       [     4.  ,     16.  ,     64.  ,    256.  ,      0.  ,      5.  ],
       [     4.5 ,     20.25,     91.12,    410.06,      0.  ,      0.  ],
       [     5.  ,     25.  ,    125.  ,    625.  ,      0.  ,      5.  ],
       [     5.5 ,     30.25,    166.38,    915.06,      1.  ,      0.  ],
       [     6.  ,     36.  ,    216.  ,   1296.  ,      1.  ,      5.  ],
       [     6.5 ,     42.25,    274.62,   1785.06,      1.  ,      0.  ],
       [     7.  ,     49.  ,    343.  ,   2401.  ,      1.  ,      5.  ],
       [     7.5 ,     56.25,    421.88,   3164.06,      1.  ,      0.  ],
       [     8.  ,     64.  ,    512.  ,   4096.  ,      1.  ,      5.  ],
       [     8.5 ,     72.25,    614.12,   5220.06,      1.  ,      0.  ],
       [     9.  ,     81.  ,    729.  ,   6561.  ,      1.  ,      5.  ],
       [     9.5 ,     90.25,    857.38,   8145.06,      1.  ,      0.  ],
       [    10.  ,    100.  ,   1000.  ,  10000.  ,      1.  ,      5.  ]])
```

```
column_stack((A, (t % 1==0)*5))
```

```
array([[     1.  ,      1.  ,      1.  ,      1.  ,      0.  ,      5.  ],
       [     1.5 ,      2.25,      3.38,      5.06,      0.  ,      0.  ],
       [     2.  ,      4.  ,      8.  ,     16.  ,      0.  ,      5.  ],
       [     2.5 ,      6.25,     15.62,     39.06,      0.  ,      0.  ],
       [     3.  ,      9.  ,     27.  ,     81.  ,      0.  ,      5.  ],
       [     3.5 ,     12.25,     42.88,    150.06,      0.  ,      0.  ],
       [     4.  ,     16.  ,     64.  ,    256.  ,      0.  ,      5.  ],
       [     4.5 ,     20.25,     91.12,    410.06,      0.  ,      0.  ],
       [     5.  ,     25.  ,    125.  ,    625.  ,      0.  ,      5.  ],
       [     5.5 ,     30.25,    166.38,    915.06,      1.  ,      0.  ],
       [     6.  ,     36.  ,    216.  ,   1296.  ,      1.  ,      5.  ],
       [     6.5 ,     42.25,    274.62,   1785.06,      1.  ,      0.  ],
       [     7.  ,     49.  ,    343.  ,   2401.  ,      1.  ,      5.  ],
       [     7.5 ,     56.25,    421.88,   3164.06,      1.  ,      0.  ],
       [     8.  ,     64.  ,    512.  ,   4096.  ,      1.  ,      5.  ],
       [     8.5 ,     72.25,    614.12,   5220.06,      1.  ,      0.  ],
       [     9.  ,     81.  ,    729.  ,   6561.  ,      1.  ,      5.  ],
       [     9.5 ,     90.25,    857.38,   8145.06,      1.  ,      0.  ],
       [    10.  ,    100.  ,   1000.  ,  10000.  ,      1.  ,      5.  ]])
```

```
column_stack((A, (A[:,1] % 1==0)*5))
```

```
array([[     1.  ,      1.  ,      1.  ,      1.  ,      0.  ,      5.  ],
       [     1.5 ,      2.25,      3.38,      5.06,      0.  ,      0.  ],
       [     2.  ,      4.  ,      8.  ,     16.  ,      0.  ,      5.  ],
       [     2.5 ,      6.25,     15.62,     39.06,      0.  ,      0.  ],
       [     3.  ,      9.  ,     27.  ,     81.  ,      0.  ,      5.  ],
       [     3.5 ,     12.25,     42.88,    150.06,      0.  ,      0.  ],
       [     4.  ,     16.  ,     64.  ,    256.  ,      0.  ,      5.  ],
       [     4.5 ,     20.25,     91.12,    410.06,      0.  ,      0.  ],
       [     5.  ,     25.  ,    125.  ,    625.  ,      0.  ,      5.  ],
```

(continues on next page)

(continued from previous page)

```
       [     5.5 ,     30.25,    166.38,    915.06,      1.  ,      0.  ],
       [     6.  ,     36.  ,    216.  ,   1296.  ,      1.  ,      5.  ],
       [     6.5 ,     42.25,    274.62,   1785.06,      1.  ,      0.  ],
       [     7.  ,     49.  ,    343.  ,   2401.  ,      1.  ,      5.  ],
       [     7.5 ,     56.25,    421.88,   3164.06,      1.  ,      0.  ],
       [     8.  ,     64.  ,    512.  ,   4096.  ,      1.  ,      5.  ],
       [     8.5 ,     72.25,    614.12,   5220.06,      1.  ,      0.  ],
       [     9.  ,     81.  ,    729.  ,   6561.  ,      1.  ,      5.  ],
       [     9.5 ,     90.25,    857.38,   8145.06,      1.  ,      0.  ],
       [    10.  ,    100.  ,   1000.  ,  10000.  ,      1.  ,      5.  ]])
```

Excercise 4

```
def matrace(A):
    if A.shape[0] == A.shape[1]:  # square matrix
        return sum(A.diagonal())
    else:
        return -1
```

```
B = randn(10,2); B
```

```
array([[ 0.63495405, -0.41332203],
       [-0.7815181 , -0.13233533],
       [ 0.7135537 , -2.28650789],
       [ 1.06424592, -0.59932232],
       [-0.92327638,  1.08949032],
       [-0.3044619 , -1.41034731],
       [ 2.43510089,  1.79379768],
       [ 1.72210238, -0.07562774],
       [-1.01335264, -0.55329476],
       [-0.02516625,  0.3408787 ]])
```

```
C = rand(10,10); C
```

```
array([[ 0.50088428,  0.93300777,  0.37201697,  0.77327375,  0.46656404,
         0.62428764,  0.22893919,  0.05437433,  0.25646812,  0.61869992],
       [ 0.21738864,  0.14611412,  0.6931886 ,  0.01817277,  0.77878839,
         0.72566348,  0.04703873,  0.93371262,  0.83055894,  0.58275793],
       [ 0.4355275 ,  0.64907709,  0.74873456,  0.01012651,  0.24467361,
         0.2129306 ,  0.55225695,  0.84802033,  0.76059712,  0.4016598 ],
       [ 0.30799587,  0.27933615,  0.86267317,  0.39175961,  0.6428809 ,
         0.75737427,  0.41230321,  0.56131722,  0.69304986,  0.94859853],
       [ 0.62737424,  0.37229053,  0.75572847,  0.0568789 ,  0.94284048,
         0.97024807,  0.22325789,  0.93904313,  0.83170287,  0.65638246],
       [ 0.45016252,  0.68649037,  0.71527278,  0.16762903,  0.33766686,
         0.60118736,  0.34700568,  0.72236911,  0.70477416,  0.25416372],
       [ 0.61660476,  0.1965179 ,  0.28903584,  0.10889469,  0.76724818,
         0.68422022,  0.79335408,  0.46277956,  0.91468881,  0.67903228],
       [ 0.12111865,  0.59213305,  0.1991645 ,  0.08751303,  0.89810622,
         0.15385135,  0.90440875,  0.47300424,  0.92589573,  0.12016823],
       [ 0.04863678,  0.12421563,  0.44208702,  0.52011922,  0.4823349 ,
         0.27295407,  0.8477118 ,  0.77363518,  0.74753611,  0.01119114],
       [ 0.82594717,  0.33727314,  0.04060328,  0.28458626,  0.9967468 ,
         0.18169722,  0.10745559,  0.74661605,  0.42308565,  0.52056614]])
```

```
D = around(rand(10,10)*10); D
```

```
array([[  9.,   2.,   8.,   3.,   9.,   1.,   9.,   9.,   1.,   2.],
       [  8.,   5.,   8.,   3.,   3.,   7.,   2.,   4.,   4.,   8.],
       [  3.,   5.,   0.,   6.,   9.,   5.,  10.,   3.,   4.,   2.],
       [  3.,   9.,   9.,   9.,   2.,   1.,   8.,   3.,   6.,   1.],
       [  4.,   8.,   5.,   5.,   1.,   1.,   3.,   1.,   2.,  10.],
       [  8.,   9.,   8.,   7.,   4.,   7.,   9.,   1.,   3.,   6.],
       [  4.,   5.,   3.,   0.,   9.,   6.,   7.,   6.,   8.,   4.],
       [  3.,   6.,   6.,   2.,   1.,   7.,   2.,   5.,   4.,   1.],
       [  6.,   4.,   8.,   9.,   5.,   2.,   8.,   9.,   1.,   9.],
       [  9.,   8.,   5.,   7.,   6.,   6.,   2.,   5.,   6.,   2.]])
```

```
matrace(B), B.trace()
```

```
(-1, 0.50261872238441196)
```

```
matrace(D), D.trace()
```

```
(46.0, 46.0)
```

```
matrace(D.T), D.T.trace()
```

```
(46.0, 46.0)
```

```
matrace(inv(D)), inv(D).trace()
```

```
(1.2606176554219353, 1.2606176554219353)
```

```
F = array(((1,2,3),(-1,-2,-3),(2,1,0))); F
```

```
array([[ 1,  2,  3],
       [-1, -2, -3],
       [ 2,  1,  0]])
```

```
G = array(((1,2,3),(-1,-2,-3))); G
```

```
array([[ 1,  2,  3],
       [-1, -2, -3]])
```

```
matrace(F), F.trace()  ## -1 = error or result?
```

```
(-1, -1)
```

```
matrace(G), G.trace()  ## -1 = error or result?
```

```
(-1, -1)
```

Try to use exceptions to idicate an error condition!

```
def matrace(A):
    if A.shape[0] == A.shape[1]:  # square matrix
        return sum(A.diagonal())
    else:
        raise ValueError("Not a square matrix. Sorry.")
```

```
matrace(F), F.trace() ## -1 must is result
```

```
(-1, -1)
```

```
matrace(G), G.trace()  ## error throws exception
```

```
---------------------------------------------------------------------------

ValueError                                Traceback (most recent call last)

<ipython-input-17-a14741b5c239> in <module>()
----> 1 matrace(G), G.trace()  ## error throws exception


<ipython-input-15-933595dad0e3> in matrace(A)
      3         return sum(A.diagonal())
      4     else:
----> 5         raise ValueError("Not a square matrix. Sorry.")


ValueError: Not a square matrix. Sorry.
```

```
try:
    matrace(G)
except( ValueError):
    print("Exception catched! Calculation of trace didn't work")
```

```
Exception catched! Calculation of trace didn't work
```

```
# this ipython magic command imports pylab and allows the plots to reside within the␣
↪notebook
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

The following examples are based on Florian Lhuillier's lecture on Matlab which can be found online:

http://geophysik.uni-muenchen.de/~lhuillier/teaching/AD2010/

1.6 Pylab continued . . .

1.6.1 Examples based on ". . . des données sous MATLAB"

```
set_printoptions(precision=2, suppress=True)
```

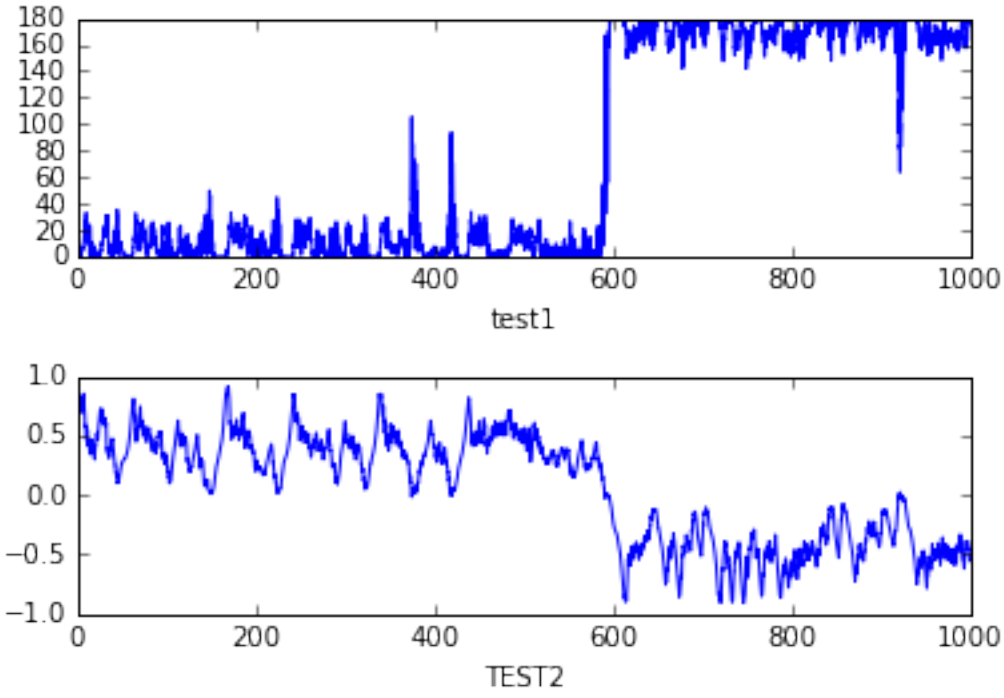
Excercise 1

```
aa = loadtxt('dipole_ref.txt')
```

```
aa.shape
```

```
(21481, 3)
```

```
subplot(211)
plot(aa[:,0], aa[:,1])
xlabel("test1")
subplot(212)
plot(aa[:,0], aa[:,2])
xlabel('TEST2')
subplots_adjust(hspace=.5)
```



Excercise 2

```
aa = loadtxt('dipole_ref.txt')
```

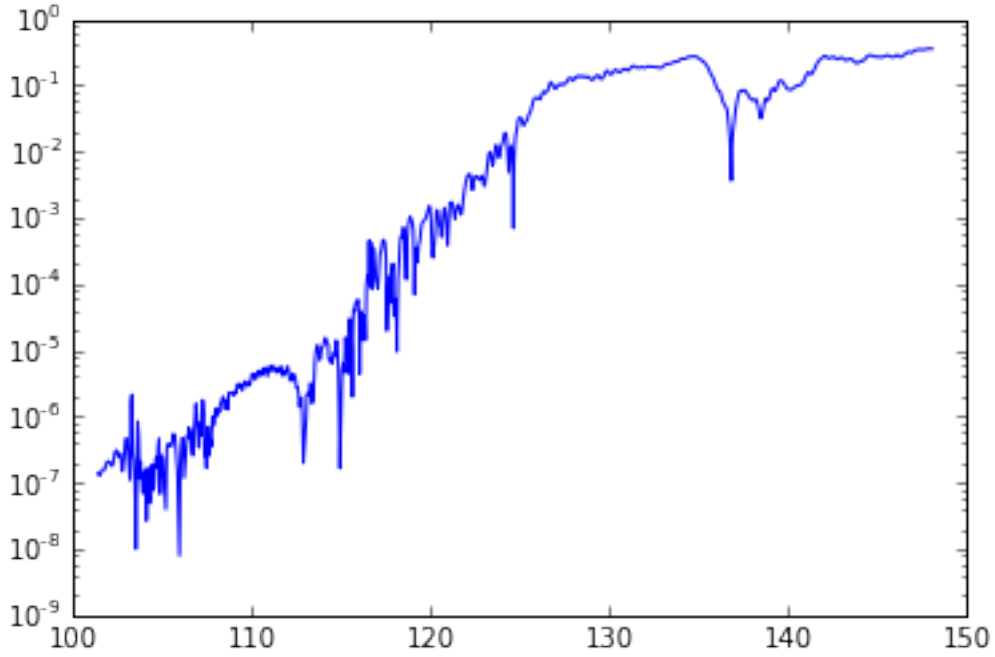
```
bb = loadtxt('dipole_pert.txt')
```

```
t = bb[:,0];
```

```
d = bb[:,2] - interp(t, aa[:,0],aa[:,2]);
```

```
semilogy(t, abs(d))
```

```
[<matplotlib.lines.Line2D at 0x7f911be2add8>]
```



Excercise 3

```
d = genfromtxt('spectre.txt', skip_header=1, dtype=None, names=('name', 'number',
↪'amplitude'))
```

```
d
```

```
array([(b'a', 1, 9.5960921), (b'b', 2, 9.4450301), (b'c', 3, 7.5000147),
       (b'd', 4, 6.7446755), (b'e', 5, 4.7442346), (b'f', 6, 4.3416987),
       (b'g', 7, 2.9904522), (b'h', 8, 2.3769902), (b'i', 9, 1.8908512),
       (b'j', 10, 1.3387981), (b'k', 11, 1.0044059),
       (b'l', 12, 0.79748122), (b'm', 13, 0.5872564),
       (b'n', 14, 0.48502449), (b'p', 15, 0.34799267),
       (b'q', 16, 0.28167023), (b'r', 17, 0.20974501),
```

(continues on next page)

(continued from previous page)

```
       (b's', 18, 0.16570928), (b't', 19, 0.12762576),
       (b'u', 20, 0.098378887)],
      dtype=[('name', 'S1'), ('number', '<i8'), ('amplitude', '<f8')])
```

```
Y=log(d['amplitude']); Y
```

```
array([ 2.26,  2.25,  2.01,  1.91,  1.56,  1.47,  1.1 ,  0.87,  0.64,
        0.29,  0.  , -0.23, -0.53, -0.72, -1.06, -1.27, -1.56, -1.8 ,
       -2.06, -2.32])
```

```
X = array((d['number'], 0*d['number']+1)).T; X
```

```
array([[ 1,  1],
       [ 2,  1],
       [ 3,  1],
       [ 4,  1],
       [ 5,  1],
       [ 6,  1],
       [ 7,  1],
       [ 8,  1],
       [ 9,  1],
       [10,  1],
       [11,  1],
       [12,  1],
       [13,  1],
       [14,  1],
       [15,  1],
       [16,  1],
       [17,  1],
       [18,  1],
       [19,  1],
       [20,  1]])
```

```
X.shape, Y.shape
```

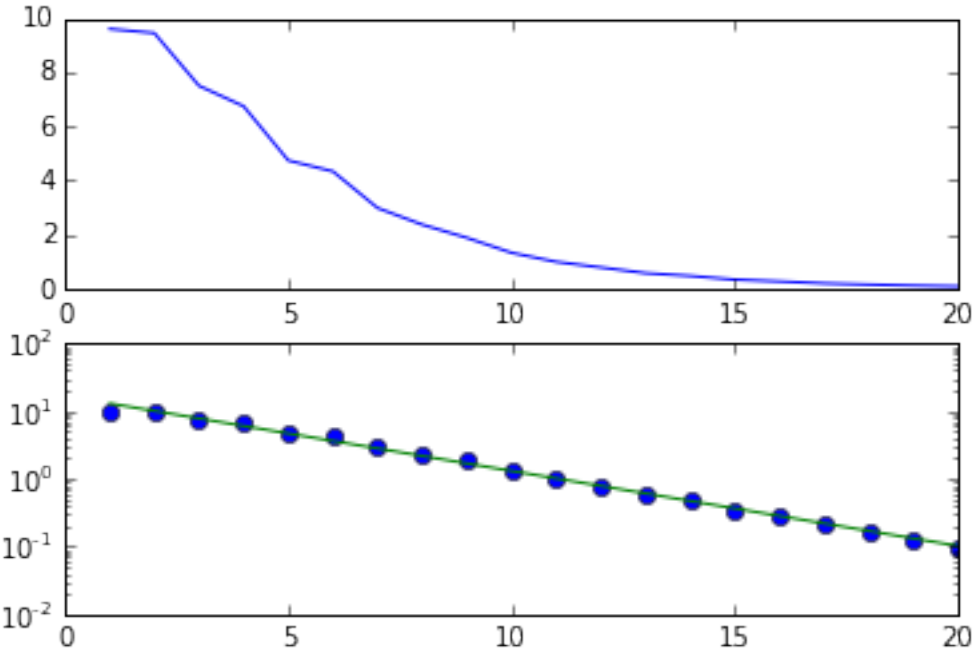
```
((20, 2), (20,))
```

```
A = lstsq(X, Y)[0]; A
```

```
array([-0.25,  2.82])
```

```
subplot(211)
plot(d['number'], d['amplitude'])
subplot(212)
semilogy(d['number'], d['amplitude'], 'o')
semilogy(d['number'], exp(X.dot(A)))
```

```
[<matplotlib.lines.Line2D at 0x7f9119b37ac8>]
```



Excercise 4

```
aa = loadtxt('anomalie.txt')
```

```
nx = len( unique(aa[:,0])); nx
```

```
21
```

```
ny = len( unique(aa[:,1])); ny
```

```
11
```

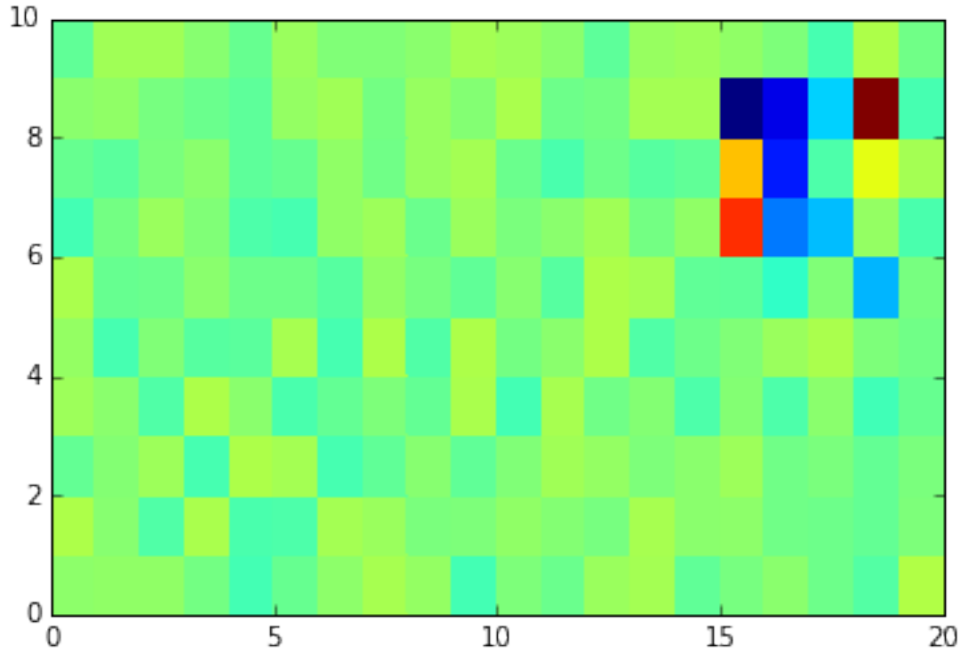
```
X=aa[:,0].reshape((nx, ny));
```

```
Y=aa[:,1].reshape((nx, ny));
```

```
data = aa[:,2].reshape((nx, ny));
```

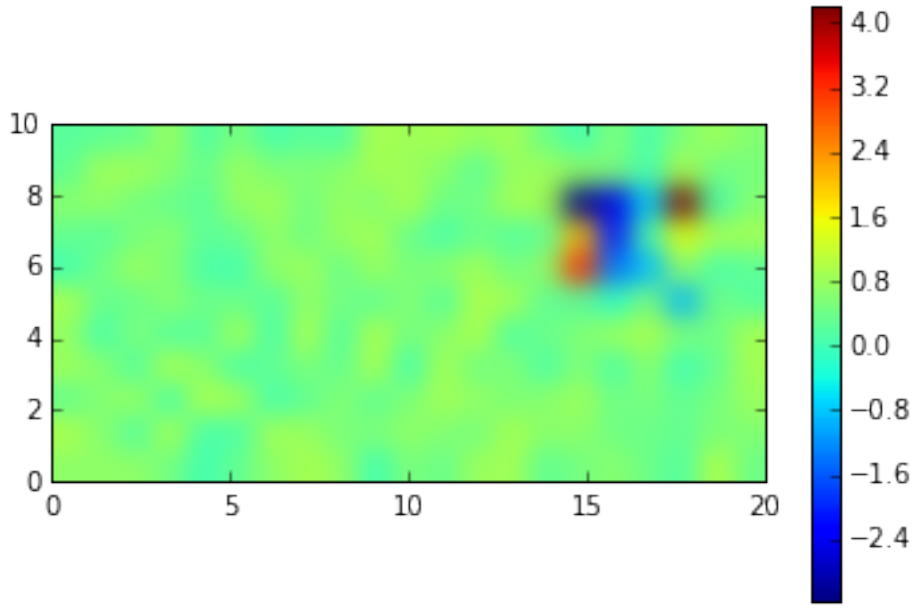
```
pcolor( X, Y, data)
```

```
<matplotlib.collections.PolyCollection at 0x7f9119cbe278>
```



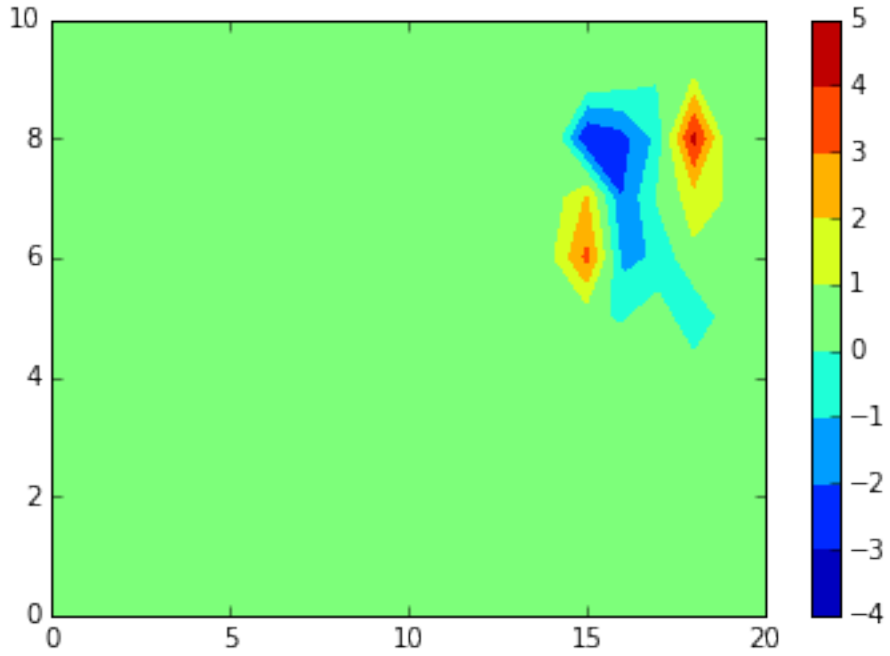

```
imshow(data.T,
           extent=[X.min(), X.max(), Y.min(), Y.max()],
           interpolation='gaussian', origin='lower')
colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f9119b30ef0>
```




```
contourf( X, Y, data)
colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f9119d93470>
```

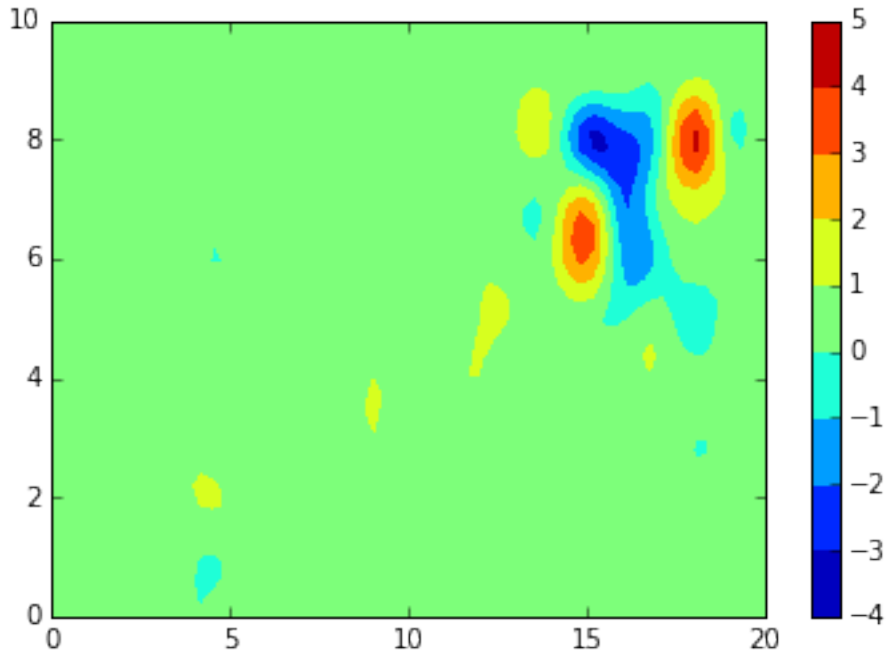



```
from scipy.ndimage import zoom
```

```
data = array(zoom(data,3)).T
```

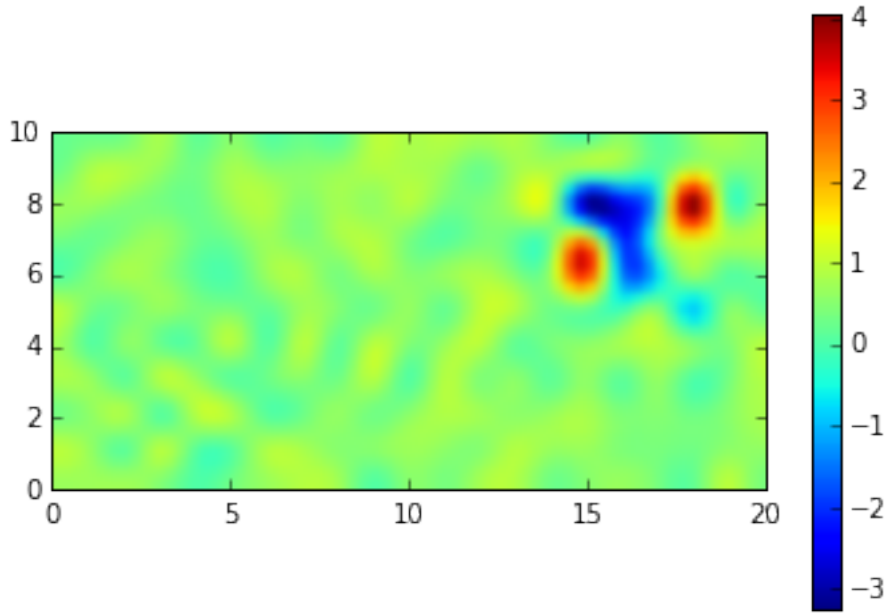
```
contourf(data, extent=[X.min(), X.max(), Y.min(), Y.max()])
colorbar()
#axis("equal")
```

```
<matplotlib.colorbar.Colorbar at 0x7f9116fb7be0>
```




```
imshow(data,
           extent=[X.min(), X.max(), Y.min(), Y.max()],
           interpolation='gaussian', origin='lower', aspect=1)
colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x7f9116ef5240>
```



1.7 Object Oriented Programming (OOP)

OOP allows you to encapsulate specific behaviour into so called classes. Instances of those classes can be used to write intuitive program code. Below is an example using classes defined in rock.py to create instances of fruits and rocks and to determine their cumulative weight on a scale. Classes can represent real world objects as in this example but also abstract things like a button in a graphical user interface or a list with special sorting algorithms.

Normally you would import the classes from the module rock like this:

```
#from rock import Rock, Sediment, MagneticMatter, MagneticSediment, Scale, Fruit
```

Alternatively you can use the magic %load command to get the whole file into this ipython notebook:

```
# %load rock.py
# object oriented programming with Python
# Michael Wack 2015

class Fruit:
    def __init__(self, weight):
        self.weight = weight

    def __repr__(self):
        return( "fruit with weight {:.2e}".format( self.weight))


class Rock:
    # class variable
    rockcounter=0


    @classmethod
    def BlueRock(cls, weight, volume):
        return cls( color="blue", weight=weight, volume=volume)


    @staticmethod
    def density( weight, volume):
        return weight / volume

    def __init__(self, color, weight, volume=11e-6):
        self.color = color
        self.weight = weight  # SI: kg
        self.volume = volume  # SI: m^3

        Rock.rockcounter += 1
        self.no = Rock.rockcounter

        print("{}. rock created".format(Rock.rockcounter))

    def calculate_density(self):
        return Rock.density( self.weight, self.volume)  # SI: kg / m^3


    def __repr__(self):
        return( "{} rock (No {}) with a density of {:.2f}".format( self.color, self.
↪no, self.calculate_density()))
```

(continues on next page)

(continued from previous page)

```
class Sediment( Rock):
    def __init__(self, color, weight, volume=11e-6, grainsize=0):
        super().__init__(color, weight, volume)

        self.grainsize = grainsize


    def double_grainsize(self):
        self.grainsize *= 2

    def __repr__(self):
        return( "{} sediment (No {}) with a density of {:.2e} and a grainsize of {:.
↪2e}".format( self.color, self.no, self.calculate_density(), self.grainsize))



class MagneticMatter(): # all volume normalized
    def __init__(self, magnetization=0, susceptibiltiy=0):
        self.magnetization = magnetization # Am^2 / m^3 = A/m
        self.susceptibility = susceptibiltiy # volume normailzed -> no units

    def induced_magnetization(self, external_field_H): # SI A/m
        return self.susceptibility * external_field_H

    def total_magnetization(self, external_field_H): # SI A/m
        return self.induced_magnetization(external_field_H) + self.magnetization



class MagneticSediment( MagneticMatter, Sediment):
    def __init__(self, color, weight, volume, grainsize=0, magnetization=0,␣
↪susceptibility=0):
        MagneticMatter.__init__( self, magnetization, susceptibility)
        Sediment.__init__( self, color, weight, volume)

    def magnetic_moment(self):
        return self.magnetization * self.volume


# class to determine weight of other object instances
class Scale():
    def __init__(self, weight_limit=100):
        self.weight_limit = weight_limit
        self.instances = []

    def put_on(self, inst):
        if inst in self.instances:
            print("{} is already on scale.".format(inst))
        else:
            self.instances.append(inst)
            print("{} placed on scale.".format(inst))

    def take_off(self, inst):
        try:
            self.instances.remove(inst)
            print("{} removed.".format(inst))
```

(continues on next page)

(continued from previous page)

```
        except KeyError:
            print("{} was not on scale.".format(inst))

    @property
    def weight(self):
        weight = 0
        for i in self.instances:
            weight += i.weight
        if weight > self.weight_limit:
            print("Scale overloaded.")

        return weight
```

```
apple = Fruit(weight=0.2)
```

```
apple
```

```
fruit with weight 2.00e-01
```

```
my_first_rock = Rock(color='red', weight=10)
```

```
1. rock created
```

```
my_first_rock
```

```
red rock (No 1) with a density of 909090.91
```

```
my_second_rock = Sediment(color='yellow', weight=12)
```

```
2. rock created
```

```
blue_rock = Rock.BlueRock(weight=3, volume = 0.1)
```

```
3. rock created
```

```
blue_rock
```

```
blue rock (No 3) with a density of 30.00
```

```
my_last_rock = MagneticSediment(color='purple', weight=1, volume=0.01,␣
↪magnetization=1e-3, susceptibility=5e-3)
```

```
4. rock created
```

```
my_last_rock.induced_magnetization(external_field_H = 40)
```

```
0.2
```

```
my_last_rock.total_magnetization(external_field_H = 40)
```

```
0.201
```

```
myscale = Scale(weight_limit = 20)
```

```
myscale
```

```
<__main__.Scale at 0x7f45b01a36d8>
```

```
myscale.put_on( blue_rock)
```

```
blue rock (No 3) with a density of 30.00 placed on scale.
```

```
myscale.weight
```

```
3
```

```
myscale.put_on( my_second_rock)
```

```
yellow sediment (No 2) with a density of 1.09e+06 and a grainsize of 0.00e+00 placed␣
↪on scale.
```

```
myscale.weight
```

```
15
```

```
myscale.put_on( my_second_rock)
```

```
yellow sediment (No 2) with a density of 1.09e+06 and a grainsize of 0.00e+00 is␣
↪already on scale.
```

```
myscale.weight
```

```
15
```

```
myscale.put_on( my_first_rock)
```

```
red rock (No 1) with a density of 909090.91 placed on scale.
```

```
myscale.weight
```

```
Scale overloaded.
```

```
25
```

```
myscale.take_off( blue_rock)
```

```
blue rock (No 3) with a density of 30.00 removed.
```

```
myscale.weight
```

```
Scale overloaded.
```

```
22
```

1.8 MagWire

As a first example how to use the modules and concepts that we have seen so far, we will develop two python classes that allow the calculation of magnetic fields generated by an arbitrary eletric current geometry. This is done by exploiting the law of Biot-Savart.

The source code including some examples is available here.

The frist class Wire is found in wire.py. It represents a continuous path for an electrical current (i.e. a wire). The path is specified as a sequence of 3D coordinates. The member property discretized_path returns the original path with linearly interpolated segments shorter than the specified discretization length.

```
__author__ = 'wack'

# part of the magwire package

# calculate magnetic fields arising from electrical current through wires of␣
↪arbitrary shape
# with the law of Biot-Savart

# written by Michael Wack
# wack@geophysik.uni-muenchen.de

# tested with python 3.4.3

from copy import deepcopy
import numpy as np
try:
    import visvis as vv
    visvis_avail = True
except ImportError:
    visvis_avail = False
    print("visvis not found.")


class Wire:
    '''
    represents an arbitrary 3D wire geometry
    '''
    def __init__(self, current=1, path=None, discretization_length=0.01):
        '''

        :param current: electrical current in Ampere used for field calculations
        :param path: geometry of the wire specified as path of n 3D (x,y,z) points in␣
↪a numpy array with dimension n x 3
                     length unit is meter
```

(continues on next page)

(continued from previous page)

```
        :param discretization_length: lenght of dL after discretization
        '''
        self.current = current
        self.path = path
        self.discretization_length = discretization_length


    @property
    def discretized_path(self):
        '''
        calculate end points of segments of discretized path
        approximate discretization lenghth is given by self.discretization_length
        elements will never be combined
        elements longer that self.dicretization_length will be divided into pieces
        :return: discretized path as m x 3 numpy array
        '''

        try:
            return self.dpath
        except AttributeError:
            pass

        self.dpath = deepcopy(self.path)
        for c in range(len(self.dpath)-2, -1, -1):
            # go backwards through all elements
            # length of element
            element = self.dpath[c+1]-self.dpath[c]
            el_len = np.linalg.norm(element)
            npts = int(np.ceil(el_len / self.discretization_length))  # number of␣
↪parts that this element should be split up into
            if npts > 1:
                # element too long -> create points between
                # length of new sub elements
                sel = el_len / float(npts)
                for d in range(npts-1, 0, -1):
                    self.dpath = np.insert(self.dpath, c+1, self.dpath[c] + element /␣
↪el_len * sel * d, axis=0)

        return self.dpath

    @property
    def IdL_r1(self):
        '''
        calculate discretized path elements dL and their center point r1
        :return: numpy array with I * dL vectors, numpy array of r1 vectors (center␣
↪point of element dL)
        '''
        npts = len(self.discretized_path)
        if npts < 2:
            print("discretized path must have at least two points")
            return

        IdL = np.array([self.discretized_path[c+1]-self.discretized_path[c] for c in␣
↪range(npts-1)]) * self.current
        r1 = np.array([(self.discretized_path[c+1]+self.discretized_path[c])*0.5 for␣
↪c in range(npts-1)])

```

(continues on next page)

(continued from previous page)

```
        return IdL, r1


    def vv_plot_path(self, discretized=True, color='r'):
        if not visvis_avail:
            print("plot path works only with visvis module")
            return

        if discretized:
            p = self.discretized_path
        else:
            p = self.path

        vv.plot(p, ms='x', mc=color, mw='2', ls='-', mew=0)


    def mpl3d_plot_path(self, discretized=True, show=True, ax=None, plt_style='-r'):

        if ax is None:
            fig = plt.figure(None)
            ax = ax3d.Axes3D(fig)

        if discretized:
            p = self.discretized_path
        else:
            p = self.path

        ax.plot(p[:, 0], p[:, 1], p[:, 2], plt_style)
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        ax.set_zlabel('Z')

        # make all axes the same
        #max_a = np.array((p[:, 0], p[:, 1], p[:, 2])).max()

        #ax.set_xlim3d(min(p[:, 0]), max_a)
        #ax.set_ylim3d(min(p[:, 1]), max_a)
        #ax.set_zlim3d(min(p[:, 2]), max_a)


        if show:
            plt.show()

        return ax

    def ExtendPath(self, path):
        '''
        extends existing path by another one
        :param path: path to append
        '''
        if self.path is None:
            self.path = path
        else:
            # check if last point is identical to avoid zero length segments
            if self.path[-1] == path[0]:
                self.path=np.append(self.path, path[1:], axis=1)
            else:
```

(continues on next page)

(continued from previous page)

```
                self.path=np.append(self.path, path, axis=1)

    def Translate(self, xyz):
        '''
        move the wire in space
        :param xyz: 3 component vector that describes translation in x,y and z direction
        '''
        if self.path is not None:
            self.path += np.array(xyz)

        return self

    def Rotate(self, axis=(1,0,0), deg=0):
        '''
        rotate wire around given axis by deg degrees
        :param axis: axis of rotation
        :param deg: angle
        '''
        if self.path is not None:
            n = axis
            ca = np.cos(np.radians(deg))
            sa = np.sin(np.radians(deg))
            R = np.array([[n[0]**2*(1-ca)+ca, n[0]*n[1]*(1-ca)-n[2]*sa, n[0]*n[2]*(1-ca)+n[1]*sa],
                          [n[1]*n[0]*(1-ca)+n[2]*sa, n[1]**2*(1-ca)+ca, n[1]*n[2]*(1-ca)-n[0]*sa],
                          [n[2]*n[0]*(1-ca)-n[1]*sa, n[2]*n[1]*(1-ca)+n[0]*sa, n[2]**2*(1-ca)+ca]])
            self.path = np.dot(self.path, R.T)

        return self



    # different standard paths
    @staticmethod
    def LinearPath(pt1=(0, 0, 0), pt2=(0, 0, 1)):
        return np.array([pt1, pt2]).T

    @staticmethod
    def RectangularPath(dx=0.1, dy=0.2):
        dx2 = dx/2.0; dy2 = dy/2.0
        return np.array([[dx2, dy2, 0], [dx2, -dy2, 0], [-dx2, -dy2, 0], [-dx2, dy2, 0], [dx2, dy2, 0]]).T

    @staticmethod
    def CircularPath(radius=0.1, pts=20):
        return Wire.EllipticalPath(rx=radius, ry=radius, pts=pts)

    @staticmethod
    def SinusoidalCircularPath(radius=0.1, amplitude=0.01, frequency=10, pts=100):
        t = np.linspace(0, 2 * np.pi, pts)
        return np.array([radius * np.sin(t), radius * np.cos(t), amplitude * np.cos(frequency*t)]).T

    @staticmethod
```

(continues on next page)

(continued from previous page)

```
    def EllipticalPath(rx=0.1, ry=0.2, pts=20):
        t = np.linspace(0, 2 * np.pi, pts)
        return np.array([rx * np.sin(t), ry * np.cos(t), 0]).T

    @staticmethod
    def SolenoidPath(radius=0.1, pitch=0.01, turns=30, pts_per_turn=20):
        return Wire.EllipticalSolenoidPath(rx=radius, ry=radius, pitch=pitch,␣
↪turns=turns, pts_per_turn=pts_per_turn)

    @staticmethod
    def EllipticalSolenoidPath(rx=0.1, ry=0.2, pitch=0.01, turns=30, pts_per_turn=20):
        t = np.linspace(0, 2 * np.pi * turns, pts_per_turn * turns)
        return np.array([rx * np.sin(t), ry * np.cos(t), t / (2 * np.pi) * pitch]).T

```

The second class BiotSavart performs the actual calculations. An arbitrary number Wire class instances can be added to define the current distribution in space. The member function CalculateB accepts a list of 3D coordinates and returns a list of 3D vectors corresponding to the B field at those points

```
__author__ = 'wack'

# part of the magwire package

# calculate magnetic fields arising from electrical current through wires of␣
↪arbitrary shape
# with the law of Biot-Savart

# written by Michael Wack
# wack@geophysik.uni-muenchen.de

# tested with python 3.4.3


import numpy as np
import time
#import multiprocessing as mp
from matplotlib import pyplot as plt
import mpl_toolkits.mplot3d.axes3d as ax3d

class BiotSavart:
    '''
    calculates the magnetic field generated by currents flowing through wires
    '''

    def __init__(self, wire=None):
        self.wires = []
        if wire is not None:
            self.wires.append(wire)

    def AddWire(self, wire):
        self.wires.append(wire)

    def CalculateB(self, points):
        """
        calculate magnetic field B at given points
        :param points: numpy array of n points (xyz)
```

(continues on next page)

(continued from previous page)

```
        :return: numpy array of n vectors representing the B field at given points
        """

        print("found {} wire(s).".format(len(self.wires)))
        c = 0
        # generate list of IdL and r1 vectors from all wires
        for w in self.wires:
            c += 1
            _IdL, _r1 = w.IdL_r1
            print("wire {} has {} segments".format(c, len(_IdL)))
            if c == 1:
                IdL = _IdL
                r1 = _r1
            else:
                IdL = np.vstack((IdL, _IdL))
                r1 = np.vstack((r1, _r1))
        print("total number of segments: {}".format(len(IdL)))
        print("number of field points: {}".format(len(points)))
        print("total number of calculations: {}".format(len(points)*len(IdL)))

        # now we have
        # all segment vectors multiplied by the flowing current in IdL
        # and all vectors to the central points of the segments in r1

        # calculate vector B*1e7 for each point in space
        t1 = time.process_time()
        # simple list comprehension to calculate B at each point r
        B = np.array([BiotSavart._CalculateB1(r, IdL, r1) * 1e-7 for r in points])

        # multi processing
        # slower than single processing?
        #pool = mp.Pool(processes=16)
        #B = np.array([pool.apply(_CalculateB1, args=(r, IdL, r1)) for r in points])

        t2 = time.process_time()
        print("time needed for calculation: {} s".format(t2-t1))

        return B

    def vv_PlotWires(self):
        for w in self.wires:
            w.vv_plot_path()

    def mpl3d_PlotWires(self, ax):
        for w in self.wires:
            w.mpl3d_plot_path(show=False, ax=ax)




    @staticmethod
    def _CalculateB1(r, IdL, r1):
        '''
        calculate magnetic field B for one point r in space
        :param r: 3 component numpy array representing the location where B will be␣
↪calculated
        :param IdL:  all segment vectors multiplied by the flowing current
```

(continues on next page)

(continued from previous page)

```
        :param r1: all vectors to the central points of the segments
        :return: numpy array of 3 component vector of B multiplied by 1e7
        '''

        # calculate law of biot savart for all current elements at given point r
        r2 = r - r1
        r25 = np.linalg.norm(r2, axis=1)**3
        r3 = r2 / r25[:, np.newaxis]

        cr = np.cross(IdL, r3)

        # claculate sum of contributions from all current elements
        s = np.sum(cr, axis=0)

        return s
```

solenoid_demo.py demonstrates the calculation and plotting of B fields generated by a simple solenoid.

```
__author__ = 'wack'

# part of the magwire package

# calculate magnetic fields arising from electrical current through wires of arbitrary shape
# with the law of Biot-Savart

# written by Michael Wack 2015
# wack@geophysik.uni-muenchen.de

# tested with python 3.4.3

# some basic calculations for testing

import numpy as np
import matplotlib.pyplot as plt
import wire
import biotsavart


# simple solenoid
# approximated analytical solution: B = mu0 * I * n / l = 4*pi*1e-7[T*m/A] * 100[A] * 10 / 0.5[m] = 2.5mT


w = wire.Wire(path=wire.Wire.SolenoidPath(pitch=0.05, turns=10), discretization_length=0.01, current=100).Rotate(axis=(1, 0, 0), deg=90) #.Translate((0.1, 0.1, 0)).
sol = biotsavart.BiotSavart(wire=w)

resolution = 0.04
volume_corner1 = (-.2, -.8, -.2)
volume_corner2 = (.2+1e-10, .3, .2)

# matplotlib plot 2D
# create list of xy coordinates
```

(continues on next page)

(continued from previous page)

```
grid = np.mgrid[volume_corner1[0]:volume_corner2[0]:resolution, volume_
↪corner1[1]:volume_corner2[1]:resolution]

# create list of grid points
points = np.vstack(map(np.ravel, grid)).T
points = np.hstack([points, np.zeros([len(points),1])])

# calculate B field at given points
B = sol.CalculateB(points=points)


Babs = np.linalg.norm(B, axis=1)

# remove big values close to the wire
cutoff = 0.005

B[Babs > cutoff] = [np.nan,np.nan,np.nan]
#Babs[Babs > cutoff] = np.nan

for ba in B:
    print(ba)

#2d quiver
# get 2D values from one plane with Z = 0

fig = plt.figure()
ax = fig.gca()
ax.quiver(points[:, 0], points[:, 1], B[:, 0], B[:, 1], scale=.15)
X = np.unique(points[:, 0])
Y = np.unique(points[:, 1])
cs = ax.contour(X, Y, Babs.reshape([len(X), len(Y)]).T, 10)
ax.clabel(cs)
plt.xlabel('x')
plt.ylabel('y')
plt.axis('equal')
plt.show()


# matplotlib plot 3D

grid = np.mgrid[volume_corner1[0]:volume_corner2[0]:resolution*2, volume_
↪corner1[1]:volume_corner2[1]:resolution*2, volume_corner1[2]:volume_
↪corner2[2]:resolution*2]

# create list of grid points
points = np.vstack(map(np.ravel, grid)).T

# calculate B field at given points
B = sol.CalculateB(points=points)

Babs = np.linalg.norm(B, axis=1)

fig = plt.figure()
# 3d quiver
ax = fig.gca(projection='3d')
sol.mpl3d_PlotWires(ax)
ax.quiver(points[:, 0], points[:, 1], points[:, 2], B[:, 0], B[:, 1], B[:, 2],␣
↪length=0.04)
```

(continues on next page)

(continued from previous page)

```
plt.show()
```

1.9 SymPy

SymPy (http://www.sympy.org/) is a Python module for symbolic mathematics. It's similiar to commercial computer algebra systems like Maple or Mathematica.

1.9.1 Usage

Web service

SymPy live http://live.sympy.org/

SymPy gamma http://sympygamma.com/

Python Module

SymPy can be installed, imported and used like any other regular Python module.

Output can be done as nicely formatted LaTeX. This is a great way to get more complicated formulae into your manuscript insted of hassling with nested LaTeX commands. Right click on the outputted formula in any ipython notebook or the above mentioned web services to extract the corresponding LaTeX commands.

1.9.2 Basic Examples

```
%matplotlib inline
from sympy import *
```

```
Value error parsing header in AFM: b'UnderlinePosition' b'-133rUnderlineThickness␣
↪20rVersion 7.000'
```

init_printing() activates nicely formatted LaTeX output within an ipython notebook.

```
init_printing()
```

Symbols

Mathematical variables have to be defined as symbols before usage

```
x = Symbol('x')
type(x)
```

```
sympy.core.symbol.Symbol
```

```
(pi + x)**2
```

$$(x + \pi)^2$$

A short form of Symbol is S. Both can be used to define serveral symbols at once.

```
x,y,z = S('x,y,z')
```

```
x**2+y**2+z**2
```

$$x^2 + y^2 + z^2$$

Fractions

Fractions of nubers must be defined with Rational. Otherwise the numbers will be evaluated as a Python expression resulting in a regular Python floating point number.

```
4/7 # regular python division
```

$$0.5714285714285714$$

```
r1 = Rational(4,7) # sympy rational
r1
```

$$\frac{4}{7}$$

```
r2 = x/y # since x and y are already sympy symbols no explicit Rational definition is␣
↪required
r2
```

$$\frac{x}{y}$$

```
r1*r2+1
```

$$\frac{4x}{7y} + 1$$

Complex Numbers

The imaginary unit is noted as I in sympy.

```
1+I
```

$$1 + i$$

```
I**2
```

$$-1$$

```
simplify((1+I)**2)
```

$$2i$$

Numeric evaluation

```
r1.evalf(100)
```

$$0.5714285714285714285714285714285714285714285714285714285714285714285714285714285714285714285714285714$$

```
pi.evalf(100)
```

$$3.141592653589793238462643383279502884197169399375105820974944592307816406286208998628034825342117068$$

Substitution

```
f1 = x**2+x**3
```

subs can be used to evaluate a function at a specific point

```
f1.subs(x, 2)
```

$$12$$

or to substitute expressions

```
a=S('a')
```

```
f1.subs(x, a)
```

$$a^3 + a^2$$

```
f2 = f1.subs(x, sqrt(a))
f2
```

$$a^{\frac{3}{2}} + a$$

```
f2.subs(a, 1)
```

$$2$$

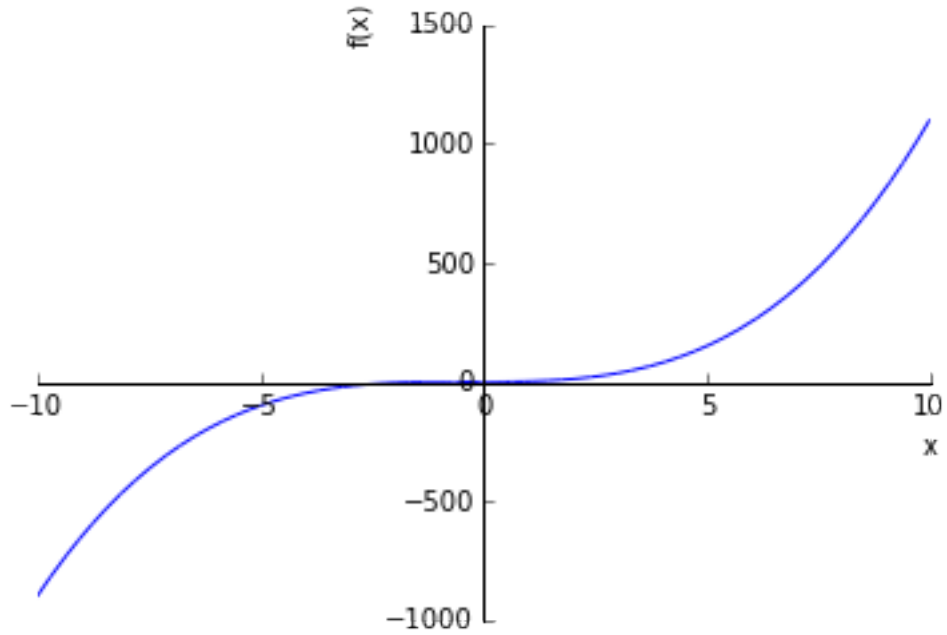
Plotting

```
from sympy.plotting import plot
```

```
f1
```

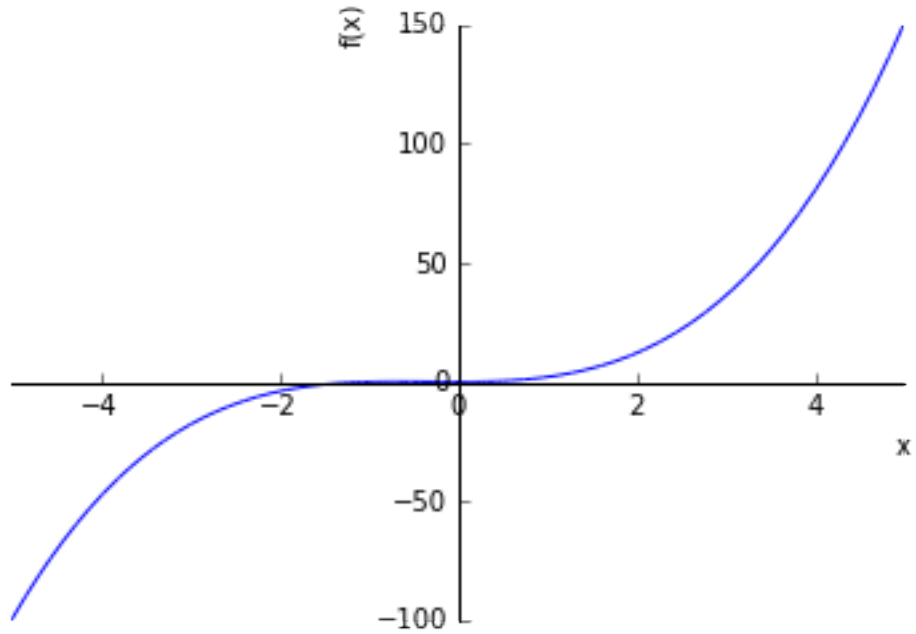
$$x^3 + x^2$$

```
plot(f1)
```




```
<sympy.plotting.plot.Plot at 0x7f1b3125a5f8>
```

```
plot(f1, (x,-5,5))
```

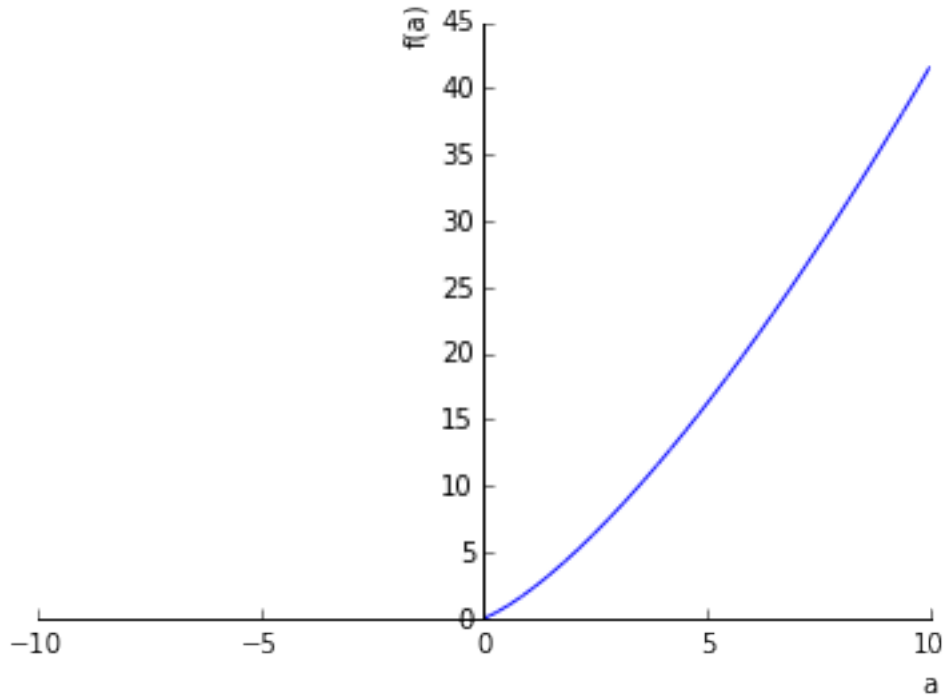



```
<sympy.plotting.plot.Plot at 0x7f1b11812e48>
```

```
f2
```

$$a^{\frac{3}{2}} + a$$

```
plot( f2)
```




```
<sympy.plotting.plot.Plot at 0x7f1b117e2898>
```

Simplification

```
x,y=S('x,y')
eq = (x**3 + x**2 - x - 1)/(x**2 + 2*x + 1)
eq
```

$$\frac{x^3 + x^2 - x - 1}{x^2 + 2x + 1}$$

```
simplify(eq)
```

$$x - 1$$

```
simplify(sin(x)**2 + cos(x)**2)
```

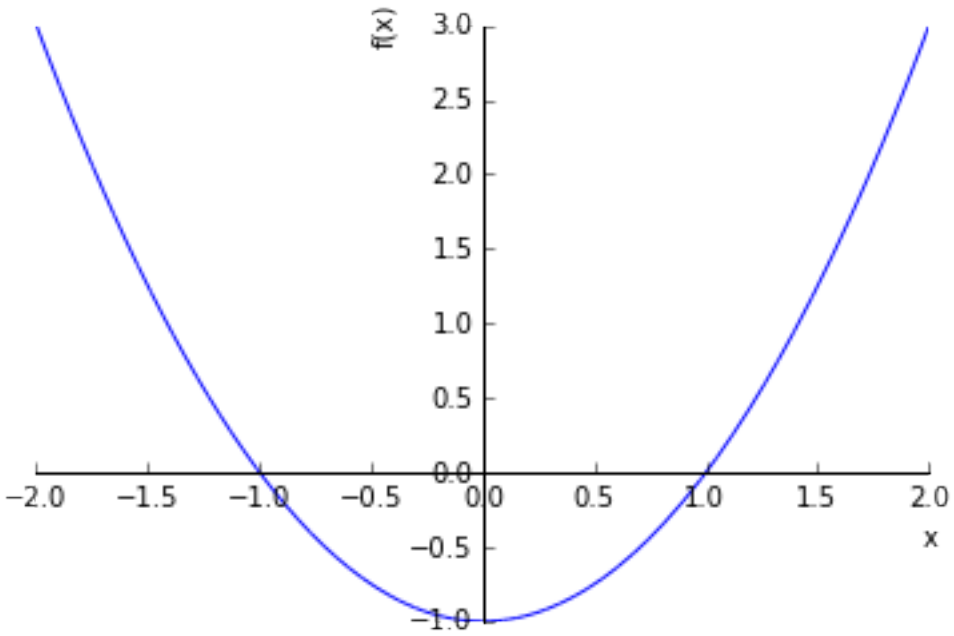
$$1$$

Solving equations

```
x = S('x')
eq = x**2 - 1
eq
```

$$x^2 - 1$$

```
plot(eq, (x, -2, +2))
```




```
<sympy.plotting.plot.Plot at 0x7f1b117665f8>
```

```
solve(eq, x)
```

$$[-1, \quad 1]$$

```
eq = 2*x*y + y**2/2
```

```
solve(eq, y)
```

$$[0, \quad -4x]$$

Differentiate

```
diff(2*x*y + y**2/2, y)
```

$$2x + y$$

Integrate

```
integrate(2*x*y + y**2/2, y)
```

$$xy^2 + \frac{y^3}{6}$$

Matrices

unity matrix

```
eye(10)
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

```
m11, m12, m21, m22 = symbols("m11, m12, m21, m22")
b1, b2 = symbols("b1, b2")
```

```
A = Matrix([[m11, m12],[m21, m22]])
A
```

$$\begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

```
b = Matrix([[b1], [b2]])
b
```

$$\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

```
A**2
```

$$\begin{bmatrix} m_{11}^2 + m_{12}m_{21} & m_{11}m_{12} + m_{12}m_{22} \\ m_{11}m_{21} + m_{21}m_{22} & m_{12}m_{21} + m_{22}^2 \end{bmatrix}$$

```
A * b
```

$$\begin{bmatrix} b_1 m_{11} + b_2 m_{12} \\ b_1 m_{21} + b_2 m_{22} \end{bmatrix}$$

```
A.det()
```

$$m_{11} m_{22} - m_{12} m_{21}$$

```
A.inv()
```

$$\begin{bmatrix} \frac{1}{m_{11}} + \frac{m_{12} m_{21}}{m_{11}^2 \left(m_{22} - \frac{m_{12} m_{21}}{m_{11}} \right)} & -\frac{m_{12}}{m_{11} \left(m_{22} - \frac{m_{12} m_{21}}{m_{11}} \right)} \\ -\frac{m_{21}}{m_{11} \left(m_{22} - \frac{m_{12} m_{21}}{m_{11}} \right)} & \frac{1}{m_{22} - \frac{m_{12} m_{21}}{m_{11}}} \end{bmatrix}$$

1.9.3 Example: Calculate determinant of matrices in a loop

```
from IPython.display import display

#b01a01
A = Matrix( [[2,2,-5],[5,4,1],[4,14,3]]) # assign matrix to variable A
B = Matrix( [[1,2,3],[0,4,5],[0,0,6]]) # assign matrix to variable B
C = Matrix( [[1,2,-1,2],[-3,5,-5,-2],[5,4,3,-4], [-4,-3,5,5]]) # assign matrix to␣
↪variable C

# show the result for each matrix
for M in (A, B, C):
    print( 'The determinant of matrix')
    display(M)
    print(' is ')
    display(M.det())
```

```
The determinant of matrix
```

$$\begin{bmatrix} 2 & 2 & -5 \\ 5 & 4 & 1 \\ 4 & 14 & 3 \end{bmatrix}$$

```
is
```

$$-296$$

```
The determinant of matrix
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & 5 \\ 0 & 0 & 6 \end{bmatrix}$$

```
is
```

$$24$$

```
The determinant of matrix
```

$$\begin{bmatrix} 1 & 2 & -1 & 2 \\ -3 & 5 & -5 & -2 \\ 5 & 4 & 3 & -4 \\ -4 & -3 & 5 & 5 \end{bmatrix}$$

```
is
```

1050

1.9.4 Example: Calculate Eigenvalues and Eigenvectors

```
a,b,c,d = S('a,b,c,d')
A = Matrix( [[a,b], [c,d]])
A
```

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

```
A.eigenvals()
```

$$\left\{ \frac{a}{2} + \frac{d}{2} - \frac{1}{2}\sqrt{a^2 - 2ad + 4bc + d^2} : 1, \quad \frac{a}{2} + \frac{d}{2} + \frac{1}{2}\sqrt{a^2 - 2ad + 4bc + d^2} : 1 \right\}$$

```
A.eigenvects()
```

$$\left[\left(\frac{a}{2} + \frac{d}{2} - \frac{1}{2}\sqrt{a^2 - 2ad + 4bc + d^2}, \quad 1, \quad \left[\begin{bmatrix} -\frac{b}{\frac{a}{2} - \frac{d}{2} + \frac{1}{2}\sqrt{a^2 - 2ad + 4bc + d^2}} \\ 1 \end{bmatrix} \right] \right), \quad \left(\frac{a}{2} + \frac{d}{2} + \frac{1}{2}\sqrt{a^2 - 2ad + 4bc + d^2}, \quad 1, \quad \left[\right.$$

```
B = A.subs(((a,1),(d,1)))
B
```

$$\begin{bmatrix} 1 & b \\ c & 1 \end{bmatrix}$$

```
B.eigenvals()
```

$$\left\{ -\sqrt{bc} + 1 : 1, \quad \sqrt{bc} + 1 : 1 \right\}$$

```
B.eigenvects()
```

$$\left[\left(-\sqrt{bc} + 1, \quad 1, \quad \left[\begin{bmatrix} -\frac{b}{\sqrt{bc}} \\ 1 \end{bmatrix} \right] \right), \quad \left(\sqrt{bc} + 1, \quad 1, \quad \left[\begin{bmatrix} \frac{b}{\sqrt{bc}} \\ 1 \end{bmatrix} \right] \right) \right]$$

1.10 PyCharm IDE

PyCharm is a state of the art IDE (Integrated Development Environment) for Python maintained by the company JetBrains (http://www.jetbrains.com).

It provides a code editor with syntax highlighting and many additional tools that help efficient software development. Layout, color schemes and keymaps are highly customizable to fit your own preferences.

Some of the most useful features are:

- Concurrent code analysis with error highlighting, code completion
- Navigation within your code: file / project structure views and direct jumping to definition of classes and functions
- Context aware refactoring (e.g. renaming a variable but not its name within a string literal)
- Integrated Python Debugger
- Version Control Systems
- Unit Testing

Official Tutorial:

https://confluence.jetbrains.com/display/PYH/Getting+Started+with+PyCharm

1.11 pandas

Pandas is a module for Python to manipulate and analyze tabular data. It allows easy reading and writing to different file formats (including MS Excel).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

1.11.1 working with Excel data

```
myexcel = pd.read_excel("pandas_test.xlsx", sheetname="mydata", skiprows=2)
```

```
myexcel
```

```
myexcel[['sample','weight']]
```

calculations with rows:

```
myexcel['density'] = myexcel['weight'] / myexcel['volume']
```

save pandas dataframe back to an excel file

```
myexcel.to_excel('new_excel_file.xlsx')
```

and a text file

```
myexcel.to_csv('new_excel_file.csv', sep='\t')
```

1.11.2 intermagnet data

Data from magnetic observatories all over the world can be downloaded here:

http://www.intermagnet.org/data-donnee/download-eng.php

we start by reading in the file downloaded from intermagnet in IAGA2002 format

```
clf = pd.read_csv("clf20160202vsec.sec", skiprows=22, sep='\s+')
```

```
clf
```

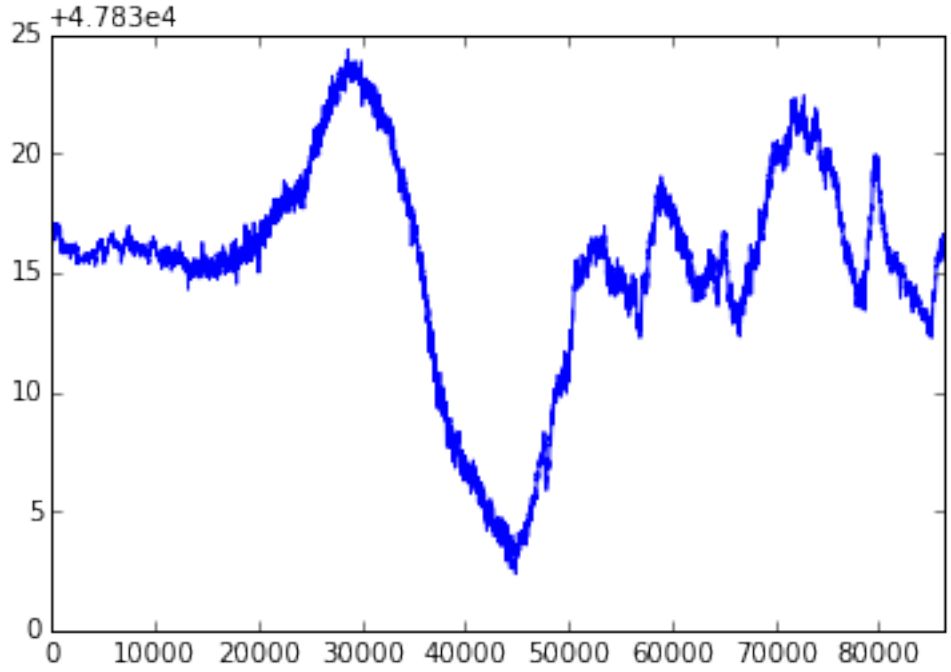
```
clf.dtypes
```

```
DATE      object
TIME      object
DOY        int64
CLFX     float64
CLFY     float64
CLFZ     float64
CLFF     float64
|        float64
dtype: object
```

plot single column

```
clf['CLFF'].plot()
```

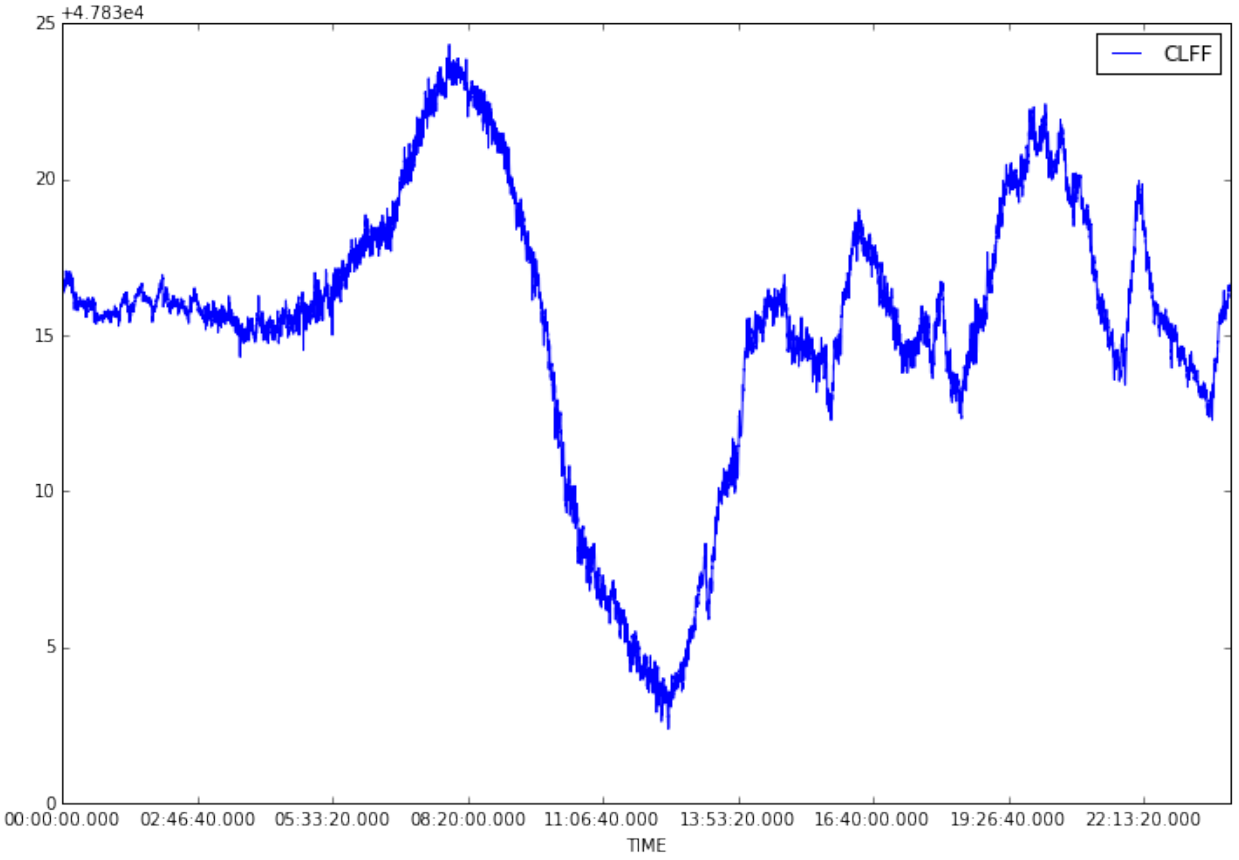
```
<matplotlib.axes._subplots.AxesSubplot at 0x97f59b0>
```



plot specific columns

```
clf.plot(x='TIME', y='CLFF', figsize=(12,8))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x93d2710>
```

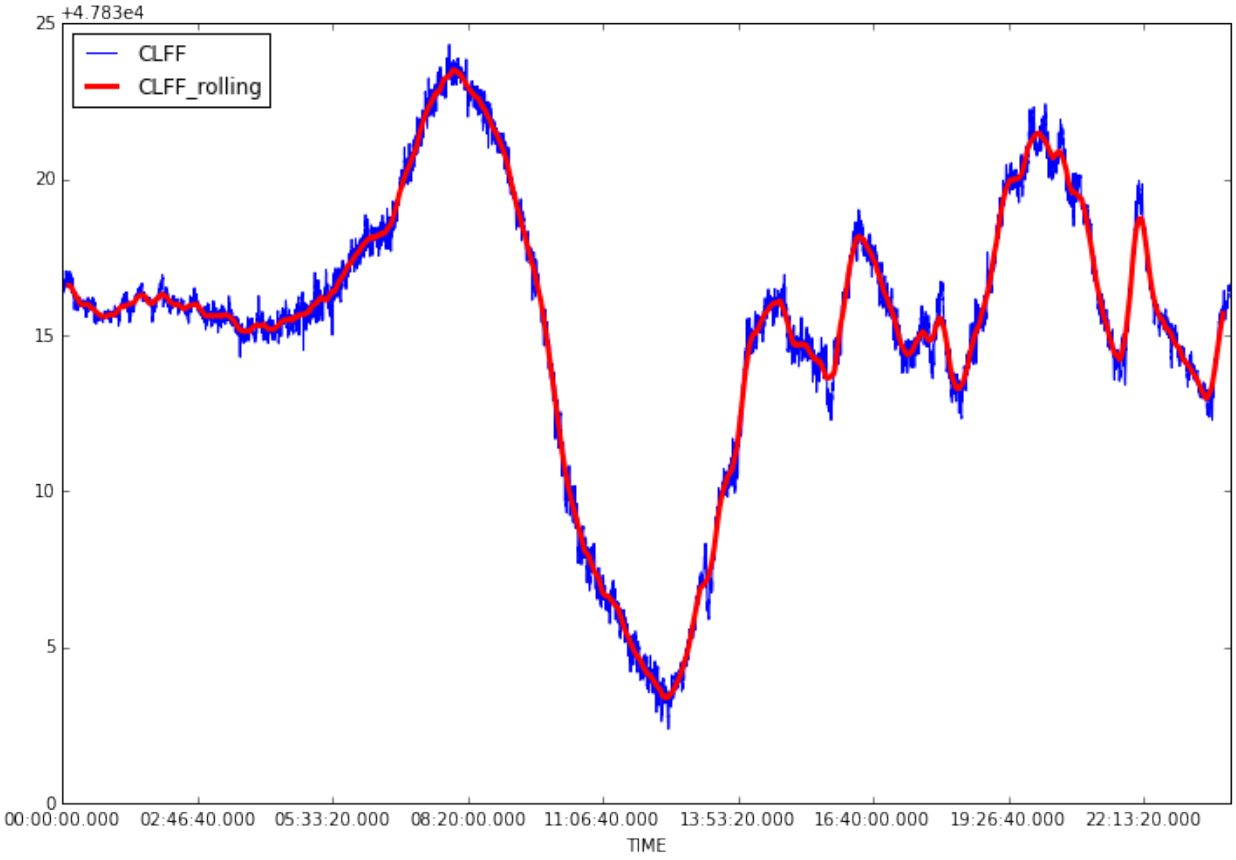


add a rolling mean

```
clf['CLFF_rolling'] = pd.rolling_mean( clf['CLFF'], window=1000, center=True)
```

```
clf.loc[490:530]
```

```
ax = clf.plot(x='TIME', y='CLFF', figsize=(12,8))
clf.plot(ax=ax, x='TIME', y='CLFF_rolling', color='r', linewidth=3)
plt.show()
```


+4.783e4
25
20
15
10
5
0
CLFF
CLFF_rolling
00:00:00.000
02:46:40.000
05:33:20.000
08:20:00.000
11:06:40.000
13:53:20.000
16:40:00.000
19:26:40.000
22:13:20.000
TIME

CHAPTER

TWO

DOWNLOADS:

All available ipython notebooks and source code can be found here

This documentation is also available in pdf format