Scientific Python Introduction

Trygve Eftestøl
Karl Skretting

Universitetet
i Stavanger

Aim of the course

- Get to know Python
  - What is Python?
    - Why use Python?
  - Basic training
    - User interface
    - Basic functions
    - Visualisation
    - Programming
    - Analysis

What is Scientific Python?

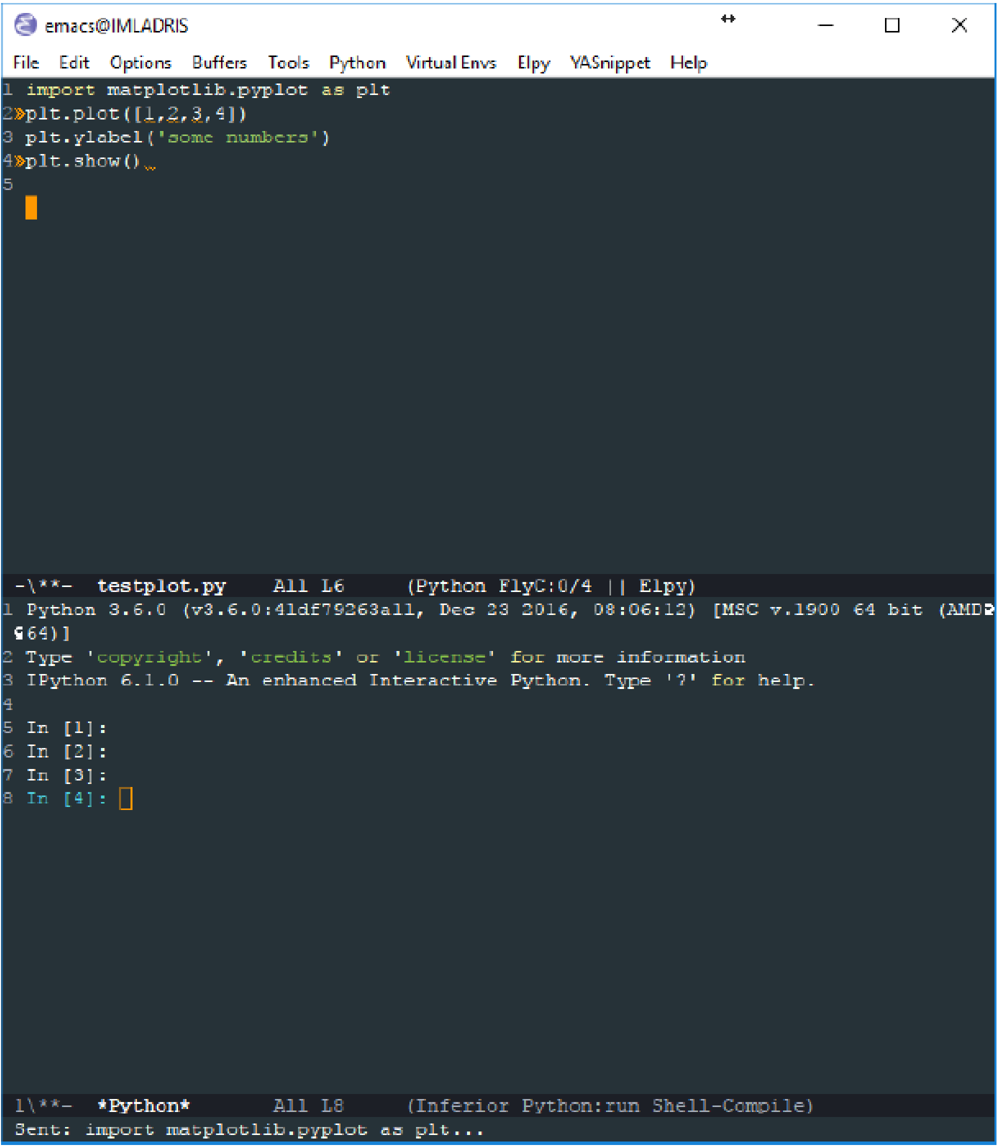
- Python
  - object oriented programming language
- Scientific
  - Add numpy, matplotlib, scipy
- Why use Python?
  - Mathematcal computations
  - Visualisation
  - Analysis
  - Used a lot for problem solving
    - Cybernetics
    - Signal processing
    - Image processing
    - Pattern recognition
    - …

Basic training

- User interface
- Basic functions
- Visualisation
- Programming
- Analysis

User interface

- Interpreter window
- Editor
- Help


emacs@IMLADRIS
File Edit Options Buffers Tools Python Virtual Envs Elpy YASnippet Help

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()

```

```
-\**-  testplot.py    All L6     (Python FlyC:0/4 || Elpy)
```

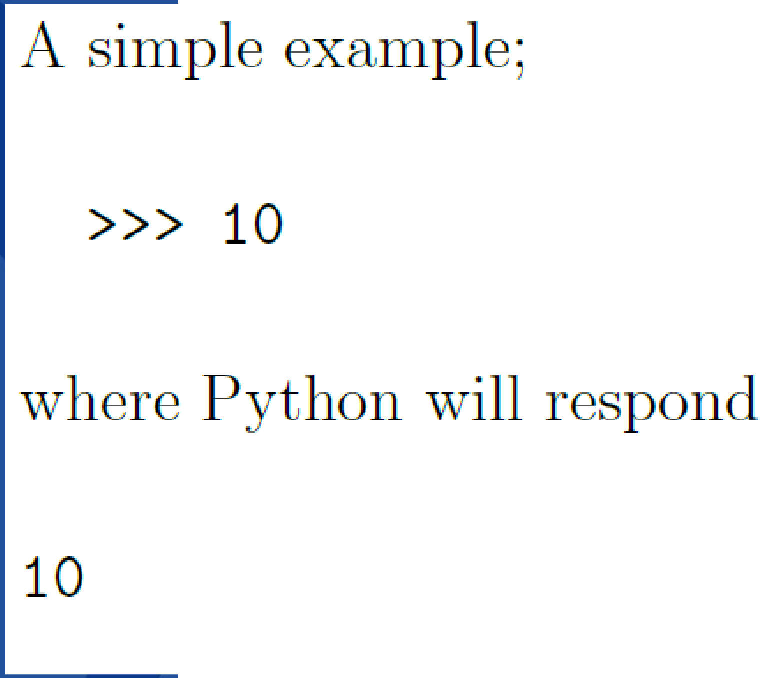
```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
In [2]:
In [3]:
In [4]:
```

```
1\**-  *Python*       All L8     (Inferior Python:run Shell-Compile)
Sent: import matplotlib.pyplot as plt...
```

User interface

- Interpreter

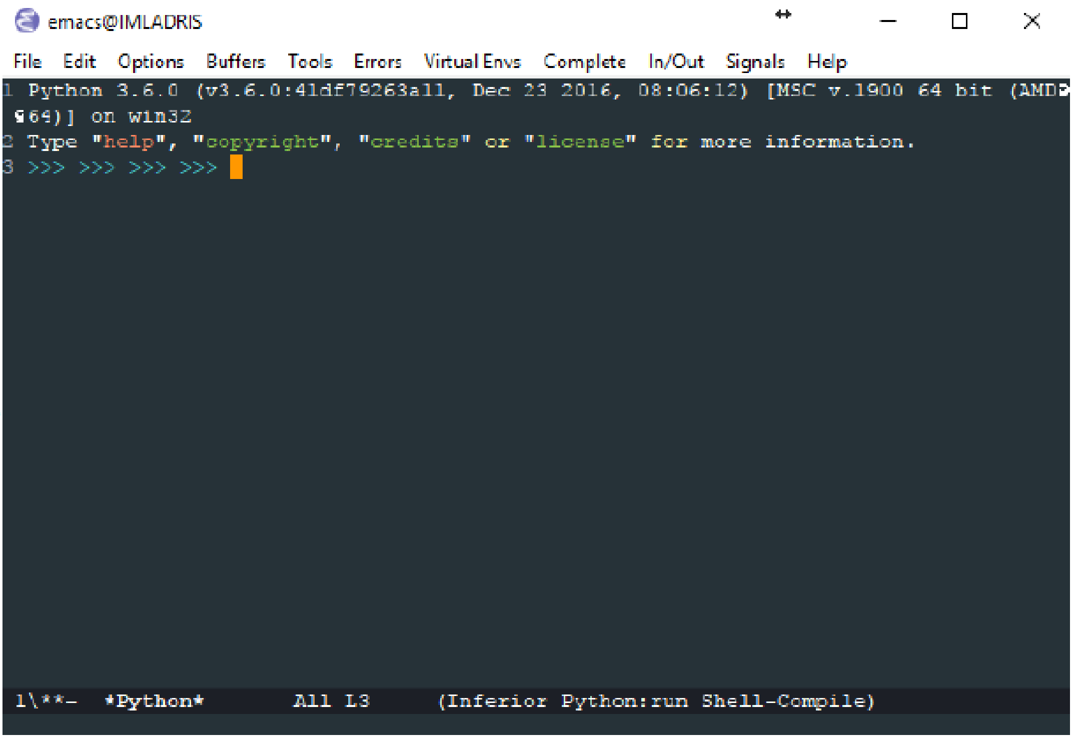


A simple example;

```
>>> 10
```

where Python will respond

```
10
```



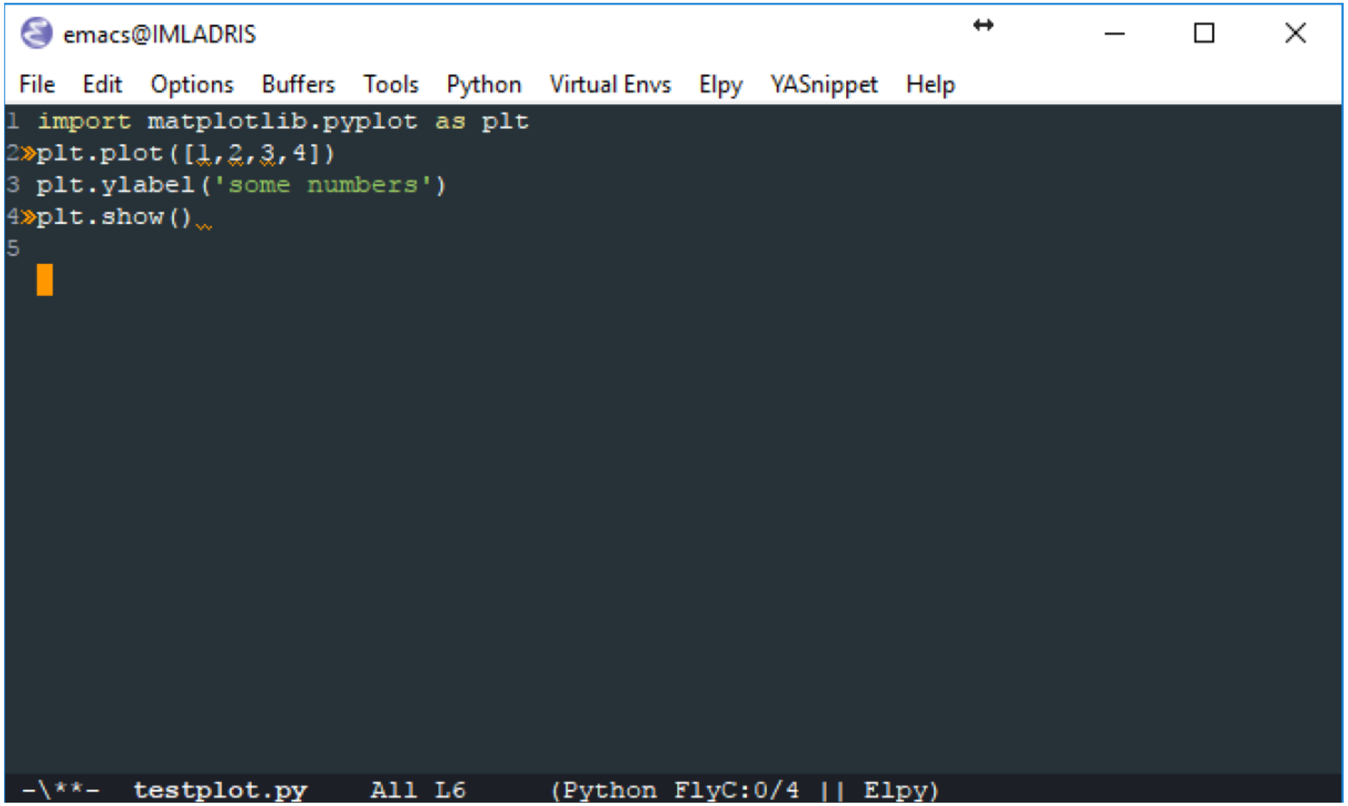
```
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 08:06:12) [MSC v.1900 64 bit (AMD
64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> >>> >>> >>> 
```

```
1\**-  *Python*       All L3     (Inferior Python:run Shell-Compile)
```

Universitetet i Stavanger

User interface

- Editor



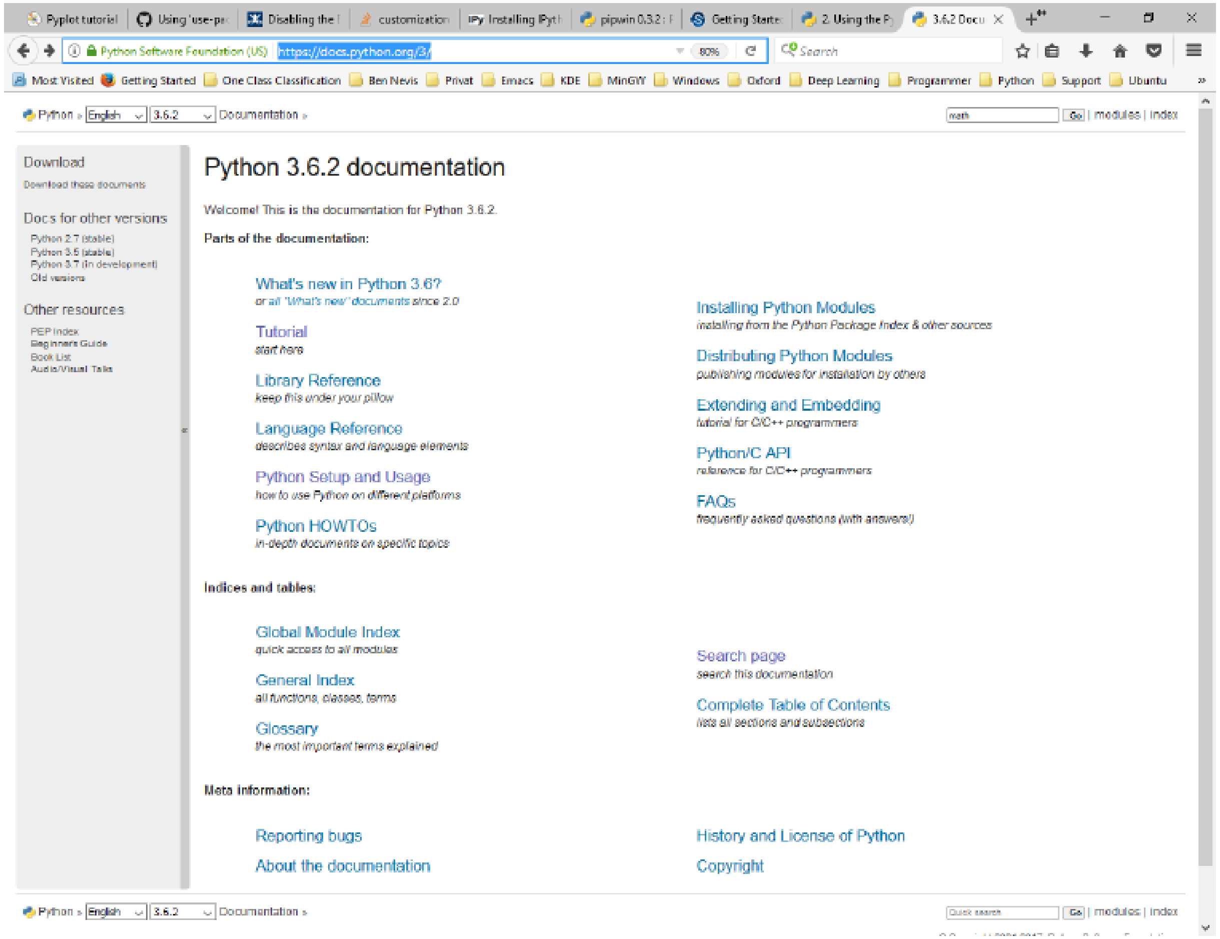
emacs@IMLADRIS
File Edit Options Buffers Tools Python Virtual Envs Elpy YASnippet Help

```
import matplotlib.pyplot as plt
plt.plot([1,2,3,4])
plt.ylabel('some numbers')
plt.show()
```

```
-\**-  testplot.py    All L6     (Python FlyC:0/4 || Elpy)
```

User interface

- Help
  - Documentation
  - Search



Download

Download these documents

Docs for other versions

Python 2.7 (stable)
Python 3.5 (stable)
Python 3.7 (in development)
Old versions

Other resources

PEP Index
Beginner's Guide
Book List
Audio/Visual Talks

Python 3.6.2 documentation

Welcome! This is the documentation for Python 3.6.2.

Parts of the documentation:

What's new in Python 3.6?
or all "What's new" documents since 2.0

Tutorial
start here

Library Reference
keep this under your pillow

Language Reference
describes syntax and language elements

Python Setup and Usage
how to use Python on different platforms

Python HOWTOs
in-depth documents on specific topics

Installing Python Modules
installing from the Python Package Index & other sources

Distributing Python Modules
publishing modules for installation by others

Extending and Embedding
tutorial for C/C++ programmers

Python/C API
reference for C/C++ programmers

FAQs
frequently asked questions (with answers!)

Indices and tables:

Global Module Index
quick access to all modules

General Index
all functions, classes, terms

Glossary
the most important terms explained

Search page
search this documentation

Complete Table of Contents
lists all sections and subsections

Meta information:

Reporting bugs

About the documentation

History and License of Python

Copyright

Python » English 3.6.2 Documentation »
Quick search Go | modules | index

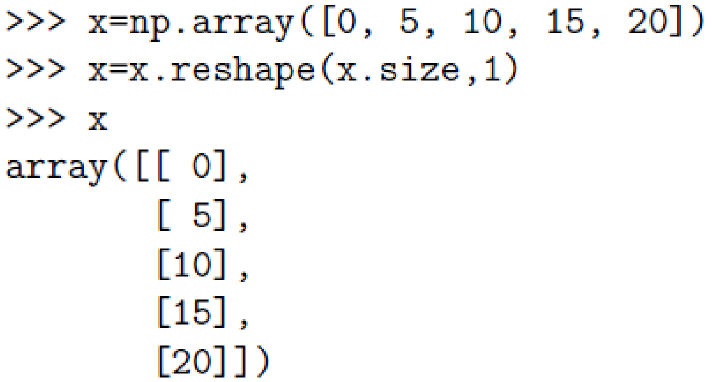
Universitetet i Stavanger

Basic functions

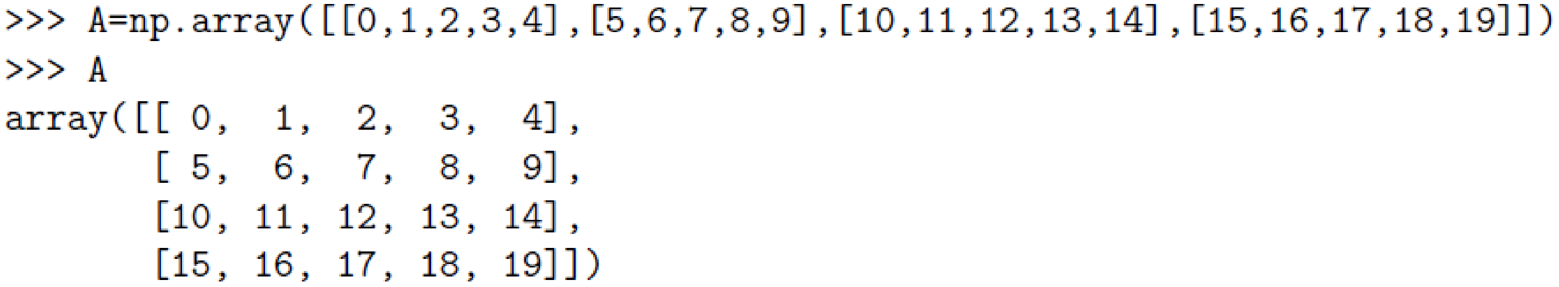
- Create matrices
- Matrix operations
- Matrix functions
- Matrix indexing
- Logical operations

Basic functions

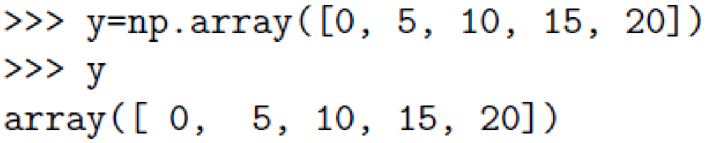
- Create matrices
  - Scalar
  - Vector
  - Matrix



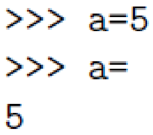
```
>>> x=np.array([0, 5, 10, 15, 20])
>>> x=x.reshape(x.size,1)
>>> x
array([[ 0],
       [ 5],
       [10],
       [15],
       [20]])
```



```
>>> A=np.array([[0,1,2,3,4],[5,6,7,8,9],[10,11,12,13,14],[15,16,17,18,19]])
>>> A
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```



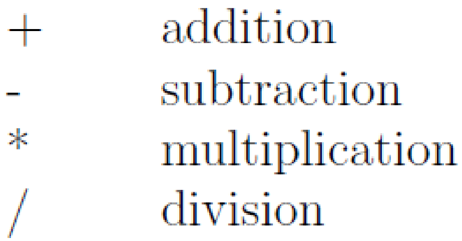
```
>>> y=np.array([0, 5, 10, 15, 20])
>>> y
array([ 0,  5, 10, 15, 20])
```



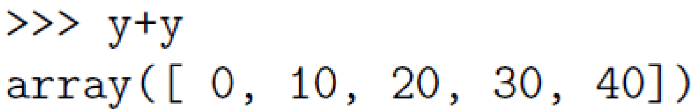
```
>>> a=5
>>> a=
5
```

Basic functions

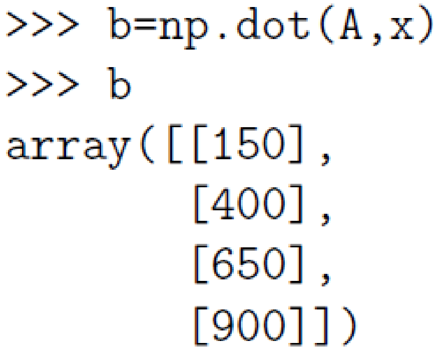
- Matrix operators



+	addition
-	subtraction
*	multiplication
/	division



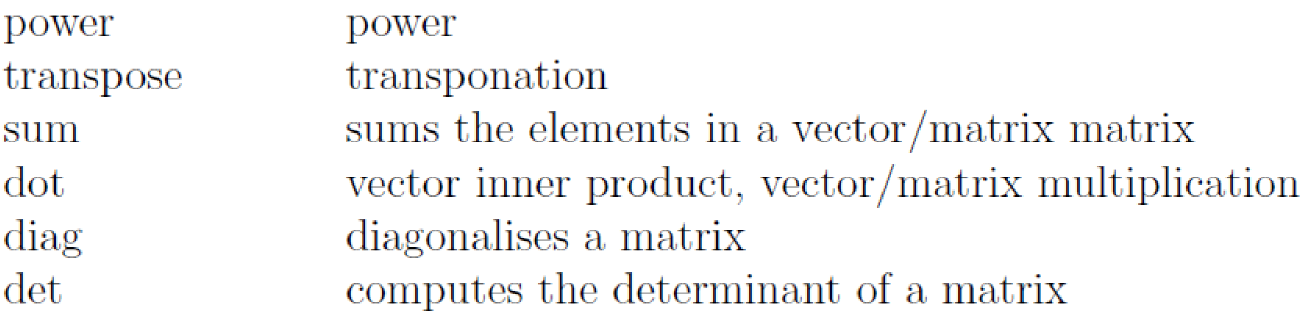
```
>>> y+y
array([ 0, 10, 20, 30, 40])
```



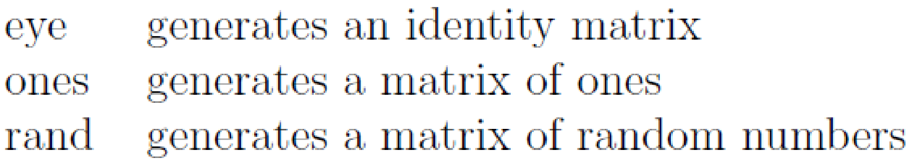
```
>>> b=np.dot(A,x)
>>> b
array([[150],
       [400],
       [650],
       [900]])
```

Basic functions

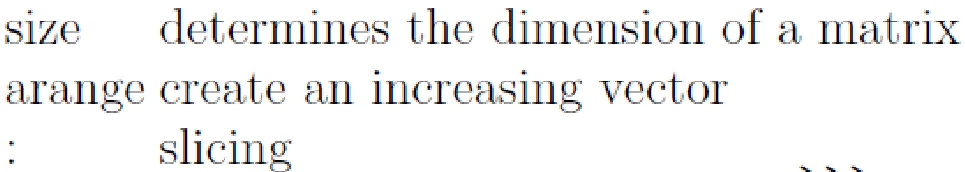
- Matrix functions



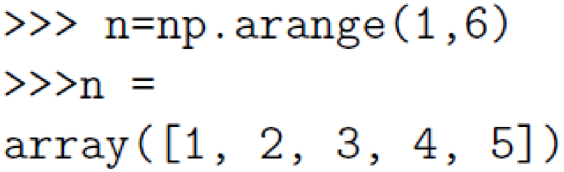
power	power
transpose	transponation
sum	sums the elements in a vector/matrix matrix
dot	vector inner product, vector/matrix multiplication
diag	diagonalises a matrix
det	computes the determinant of a matrix



eye	generates an identity matrix
ones	generates a matrix of ones
rand	generates a matrix of random numbers



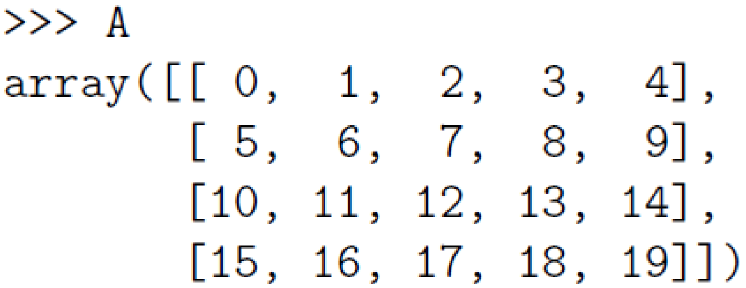
size	determines the dimension of a matrix
arange	create an increasing vector
:	slicing



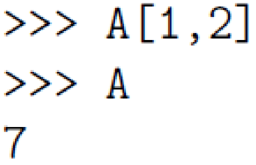
```
>>> n=np.arange(1,6)
>>>n =
array([1, 2, 3, 4, 5])
```

Basic functions

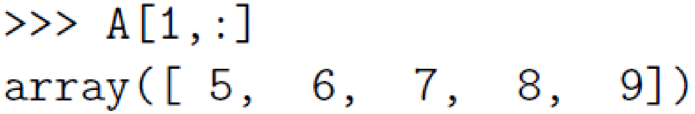
- Matrix indexing



```
>>> A
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19]])
```



```
>>> A[1,2]
>>> A
7
```



```
>>> A[1,:]
array([ 5,  6,  7,  8,  9])
```

Basic functions

- Logical operators

<	less than
>	larger than
<=	less than or equal to
>=	less than or equal to
==	equal
!=	not equal

```
>>> y
array([ 0,  5, 10, 15, 20])
```

```
>>> y>=10
array([False, False,  True,  True,  True], dtype=bool)
```

and	logical AND
or	logical OR
not	logical NOT

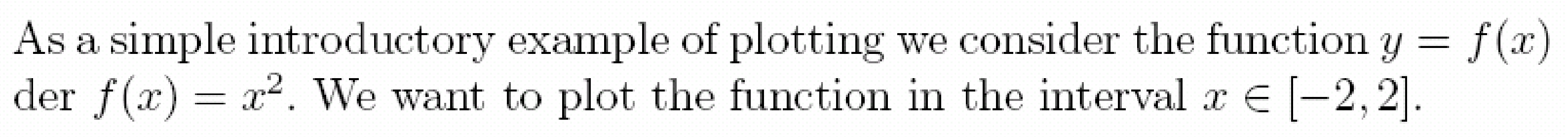
```
>>> idx=np.where(y>=10)
>>> idx
(array([2, 3, 4], dtype=int64),)
```

Visualisation

- 2D-plotting
- 3D-plotting

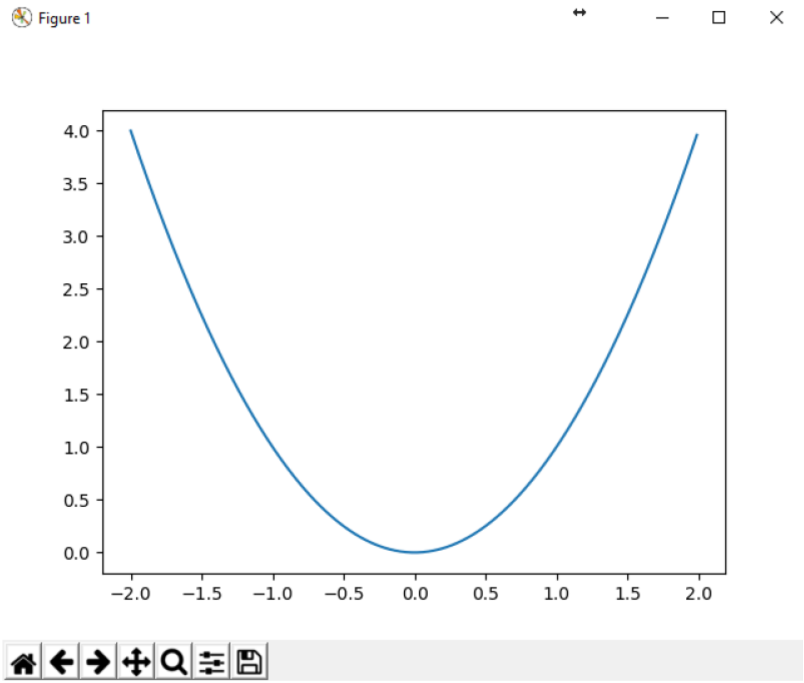
Visualisation

- 2D-plotting



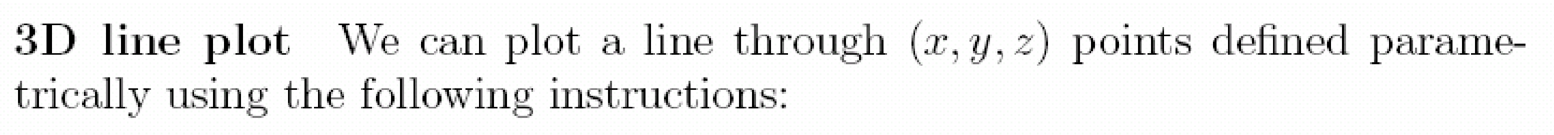
As a simple introductory example of plotting we consider the function $y = f(x)$ der $f(x) = x^2$. We want to plot the function in the interval $x \in [-2, 2]$.

```
>>> x=np.arange(-2,2,0.01)
>>> y=np.square(x)
>>> plt.plot(x,y)
>>> plt.show()
```

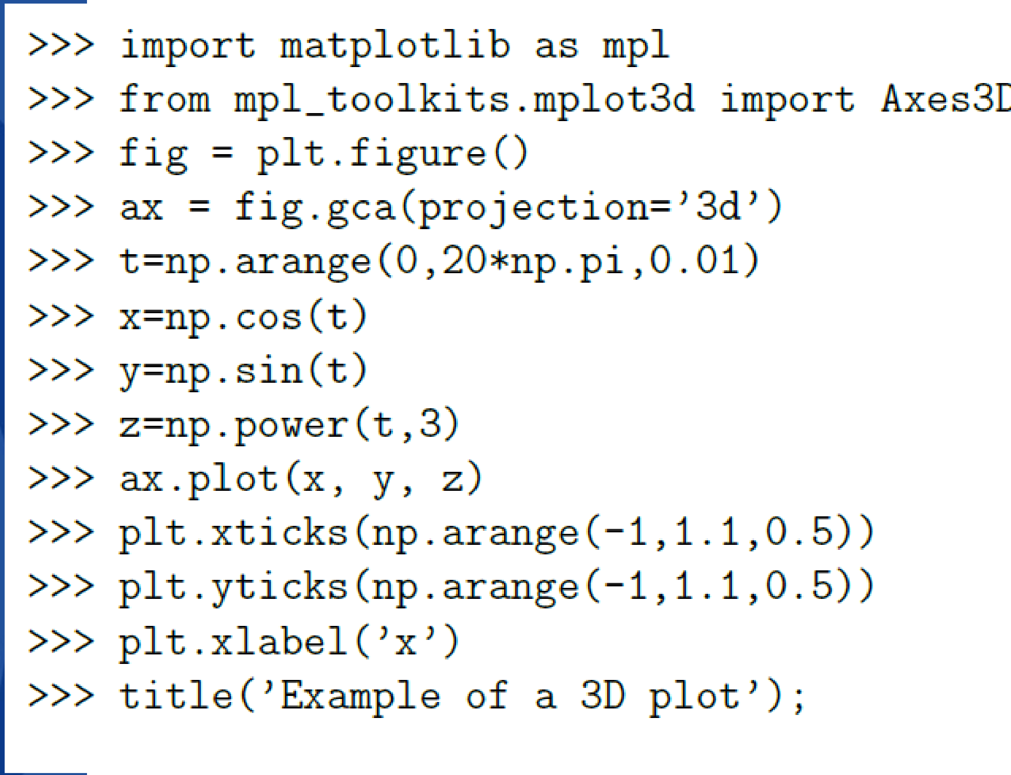


Visualisation

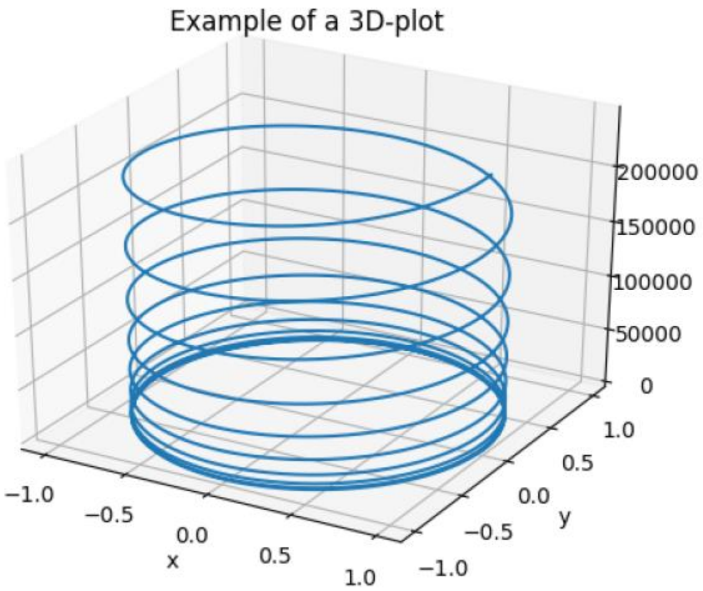
- 3D-plotting



3D line plot We can plot a line through (x, y, z) points defined parametrically using the following instructions:

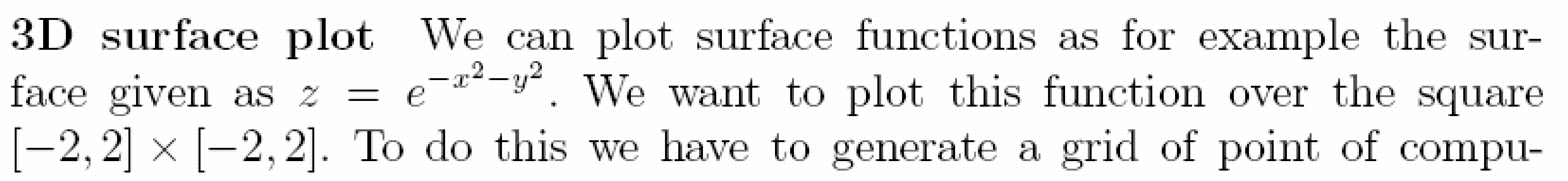


```
>>> import matplotlib as mpl
>>> from mpl_toolkits.mplot3d import Axes3D
>>> fig = plt.figure()
>>> ax = fig.gca(projection='3d')
>>> t=np.arange(0,20*np.pi,0.01)
>>> x=np.cos(t)
>>> y=np.sin(t)
>>> z=np.power(t,3)
>>> ax.plot(x, y, z)
>>> plt.xticks(np.arange(-1,1.1,0.5))
>>> plt.yticks(np.arange(-1,1.1,0.5))
>>> plt.xlabel('x')
>>> title('Example of a 3D plot');
```

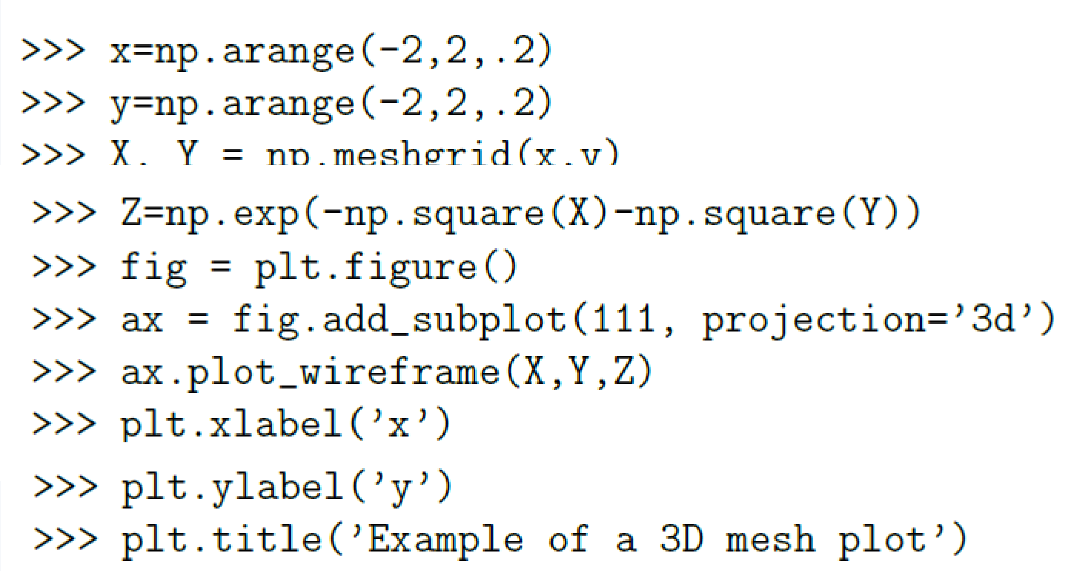


Visualisation

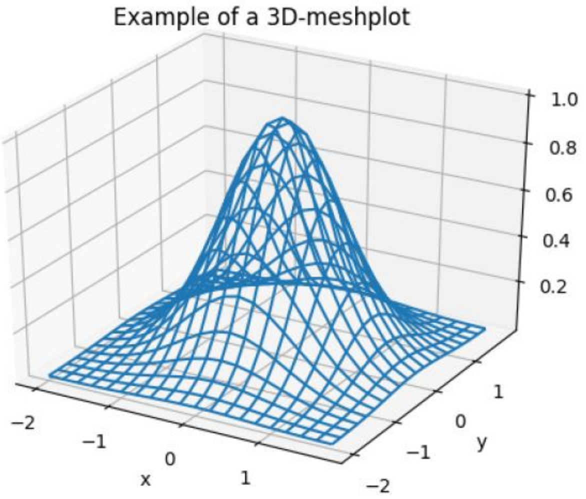
- 3D-plotting



3D surface plot We can plot surface functions as for example the surface given as $z = e^{-x^2-y^2}$. We want to plot this function over the square $[-2,2] \times [-2,2]$. To do this we have to generate a grid of point of compu-



```
>>> x=np.arange(-2,2,.2)
>>> y=np.arange(-2,2,.2)
>>> X, Y = np.meshgrid(x,y)
>>> Z=np.exp(-np.square(X)-np.square(Y))
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection=’3d’)
>>> ax.plot_wireframe(X,Y,Z)
>>> plt.xlabel(’x’)
>>> plt.ylabel(’y’)
>>> plt.title(’Example of a 3D mesh plot’)
```

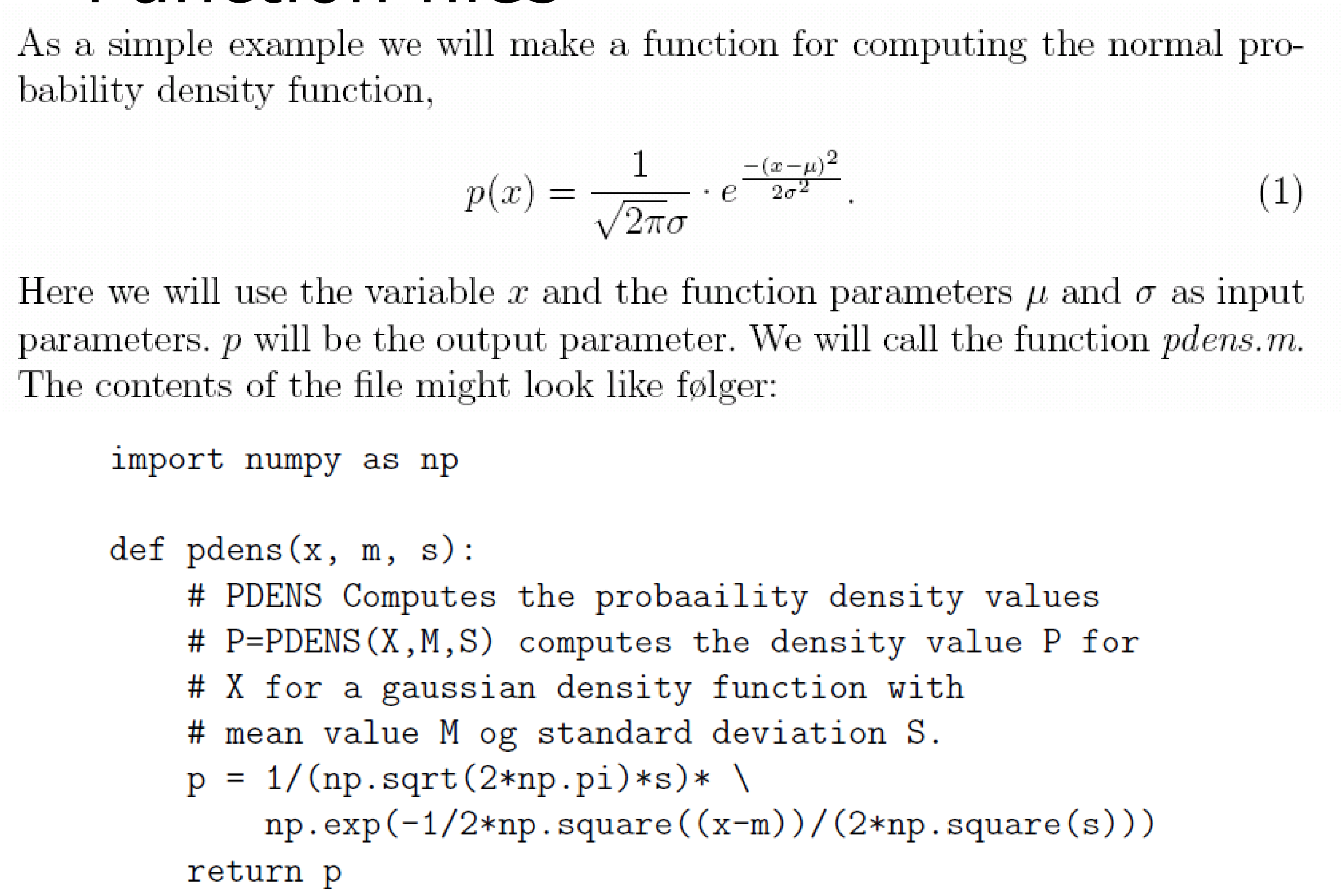


Programming

- Function files
- Control structures
  - if
  - while
  - for

Programming

- Function files



As a simple example we will make a function for computing the normal probability density function,

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} \cdot e^{\frac{-(x-\mu)^2}{2\sigma^2}}. \qquad (1)$$

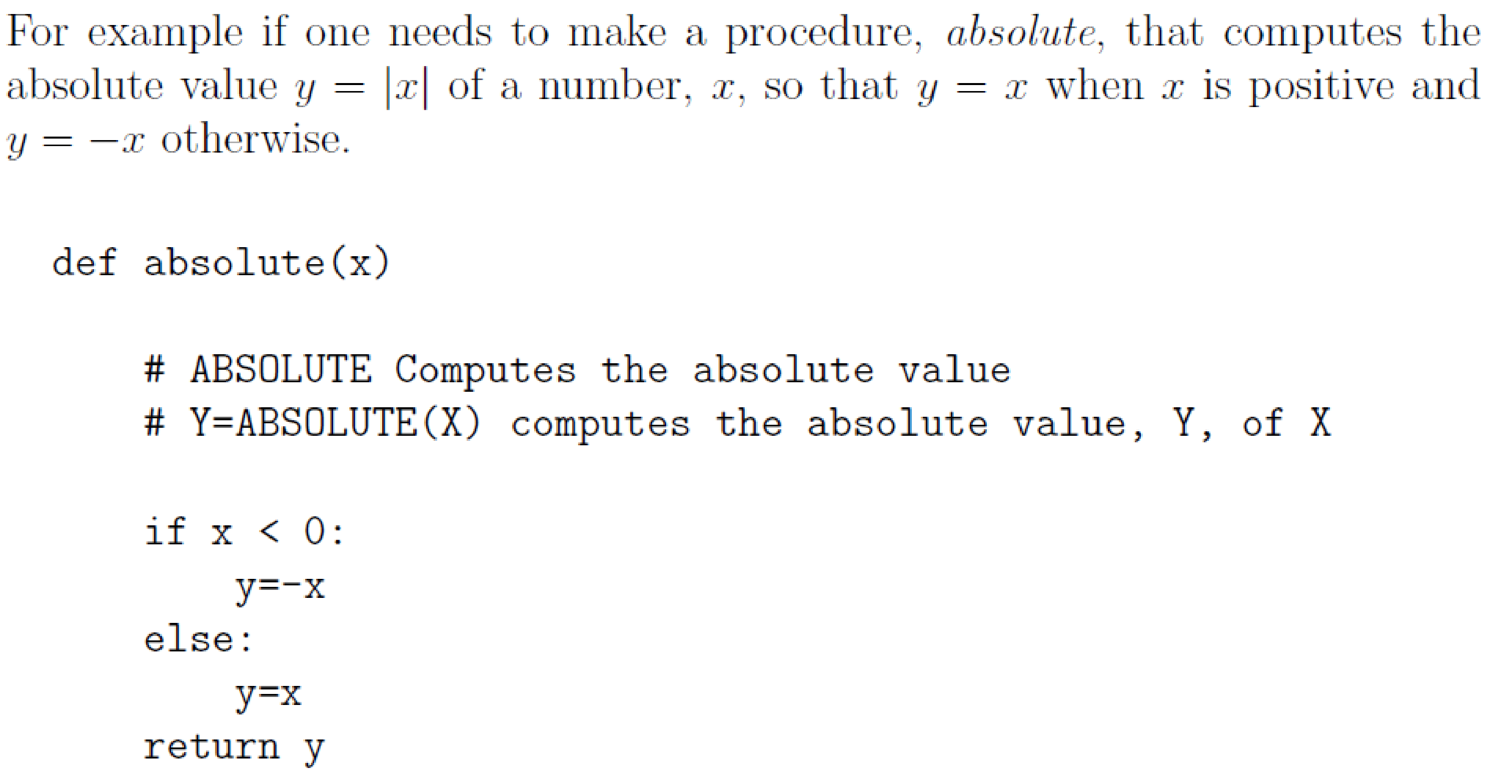
Here we will use the variable x and the function parameters μ and σ as input parameters. p will be the output parameter. We will call the function *pdens.m*. The contents of the file might look like følger:

```
import numpy as np

def pdens(x, m, s):
    # PDENS Computes the probaaility density values
    # P=PDENS(X,M,S) computes the density value P for
    # X for a gaussian density function with
    # mean value M og standard deviation S.
    p = 1/(np.sqrt(2*np.pi)*s)* \
        np.exp(-1/2*np.square((x-m))/(2*np.square(s)))
    return p
```

Programming

- Condition control using *if*



For example if one needs to make a procedure, *absolute*, that computes the absolute value $y = |x|$ of a number, x, so that $y = x$ when x is positive and $y = -x$ otherwise.

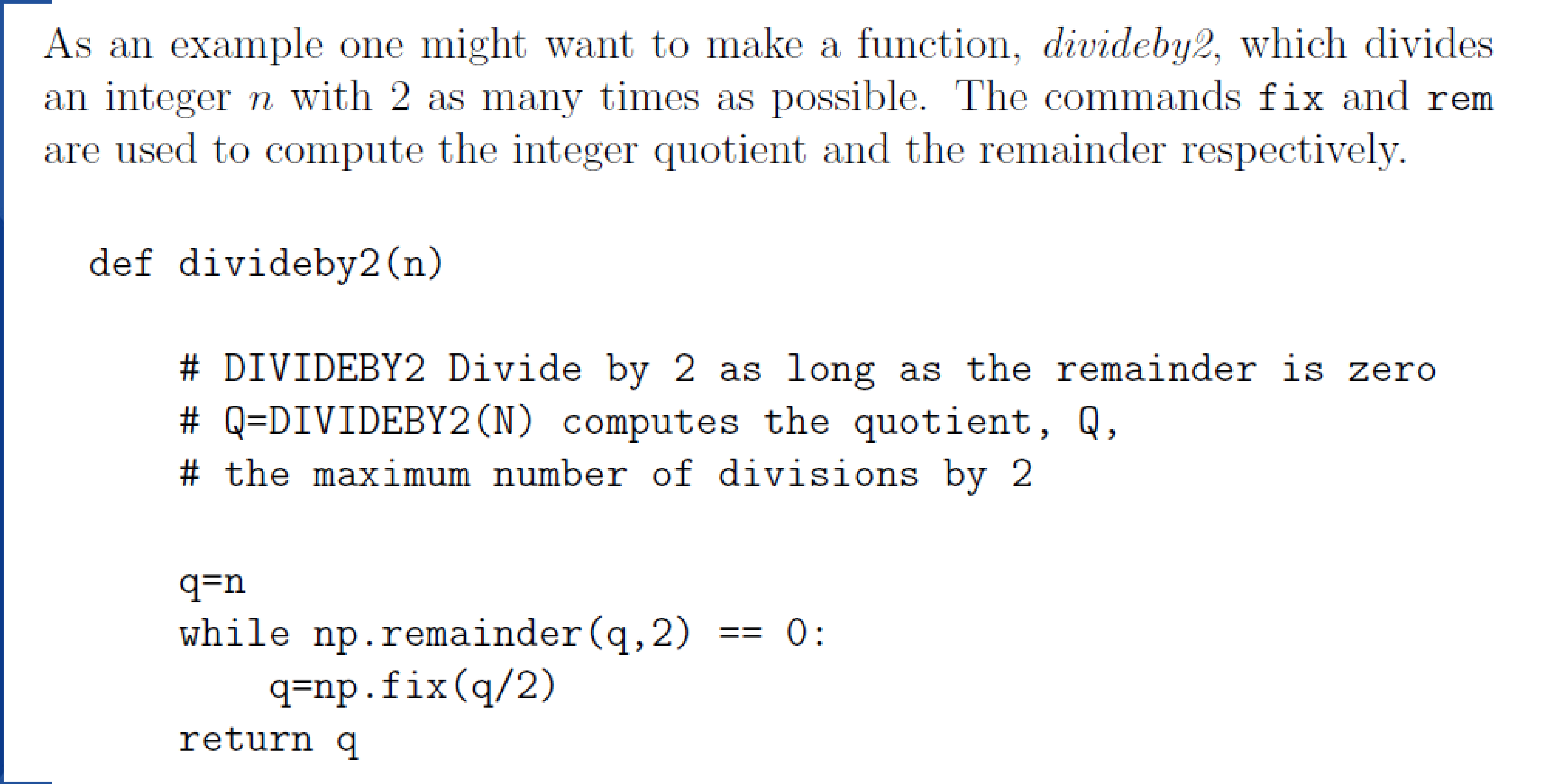
```
def absolute(x)

    # ABSOLUTE Computes the absolute value
    # Y=ABSOLUTE(X) computes the absolute value, Y, of X

    if x < 0:
        y=-x
    else:
        y=x
    return y
```

Programming

- Iterations using *while*



As an example one might want to make a function, *divideby2*, which divides an integer n with 2 as many times as possible. The commands `fix` and `rem` are used to compute the integer quotient and the remainder respectively.

```
def divideby2(n)

    # DIVIDEBY2 Divide by 2 as long as the remainder is zero
    # Q=DIVIDEBY2(N) computes the quotient, Q,
    # the maximum number of divisions by 2

    q=n
    while np.remainder(q,2) == 0:
        q=np.fix(q/2)
    return q
```

Programming

- Iterations using *for*

As an example we want to modify the function *pdens* so that you can compute the density values for more than one point of computation at a time. A possible way to do this:

```
def p=pdens2(x,m,s)

    # PDENS Computes the probaaility density values
    # P=PDENS(X,M,S) computes the density value P for
    # the values in a vector X for a gaussian density function
    # with mean value M og standard deviation S.

    N = np.shape(x)[0]
    p = np.zeros(np.shape(x))
    for n in np.arange(0, N - 1):
        p[n] = 1 / (np.sqrt(2 * np.pi) * s) * \
            np.exp(-1 / 2 * np.square((x[n] - m)) / (2 * np.square(s)))
    return p
```

Analysis

- Estimate pulse rate
  - Reading from data file
  - Plotting of signal
  - Detection of peaks
  - Problem – false detections
    - Spectral analysis
    - Filtering

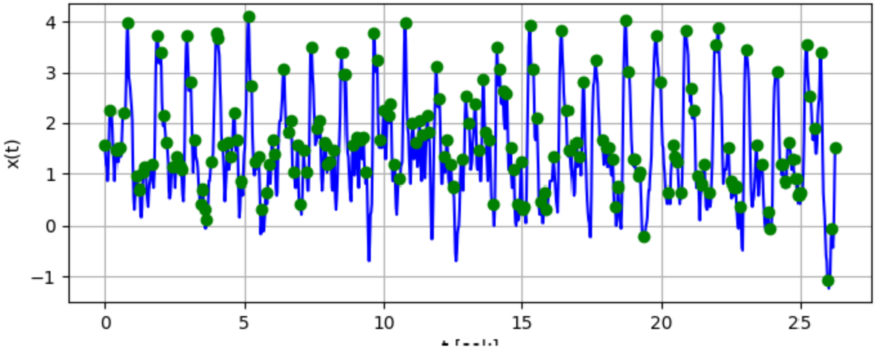
Analysis

```
>>> x=np.genfromtxt(’photopl.txt’,dtype=None,delimiter=None,skip_header=4
>>> x=x/np.std(x)
```

```
>>> fs = 40.0
>>> N = np.size(x)
>>> t = np.arange(0, N) / fs
>>> t = t.reshape(t.size, 1)
```

- Reading from data file
- Plotting of signal
- Detection of peaks

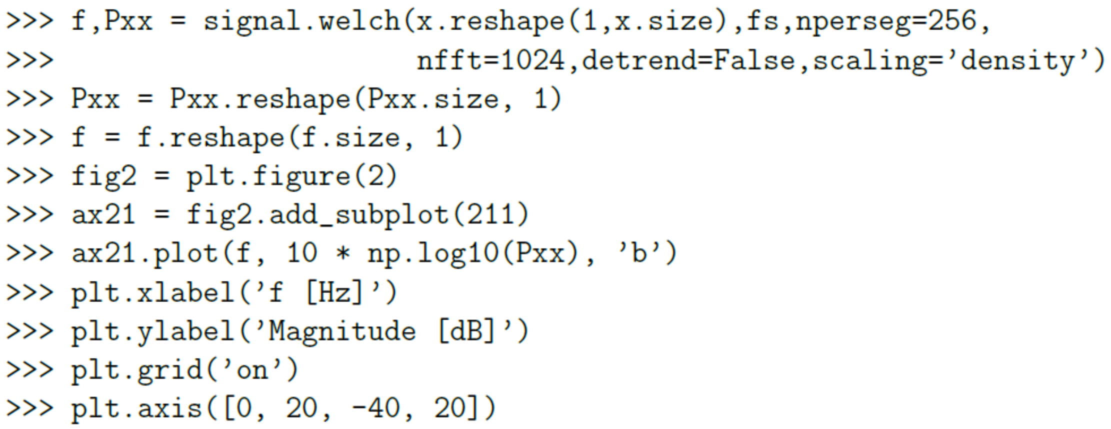
```
>>> from analyse import findpeaks
>>> ind, peaks = findpeaks(x)
```



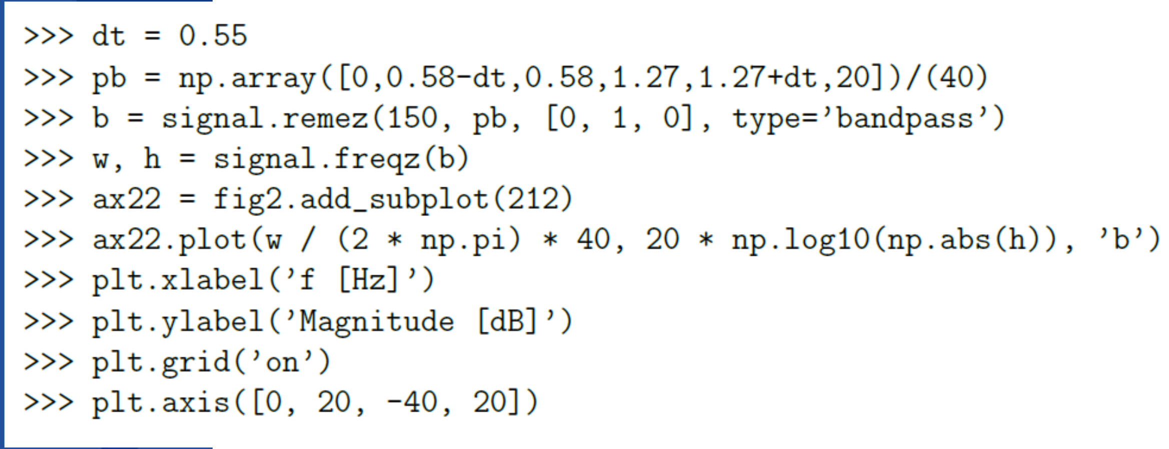
```
>>> fig1 = plt.figure(1)
>>> ax1 = plt.axes()
>>> ax1.plot(t, x, ’b’, t[ind, 0], x[ind, 0], ’go’)
>>> plt.xlabel(’t [sek]’;
>>> plt.ylabel(’x(t)’)
>>> plt.grid(’on’)
>>> fig1.set_size_inches(8, 3)
```

Analysis

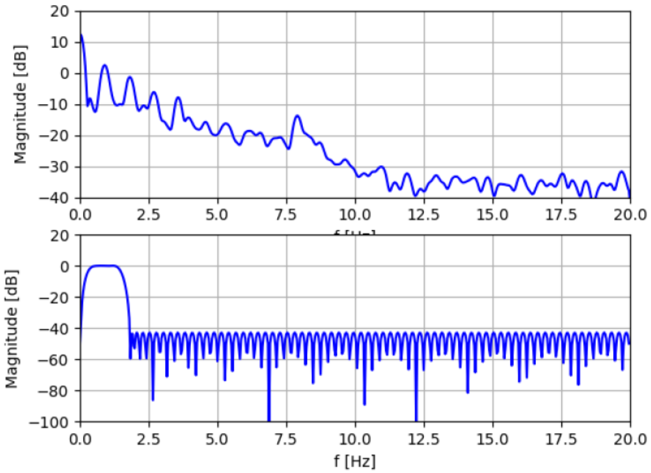
- Problem – false detections
  - Spectral analysis
  - Filtering



```
>>> f,Pxx = signal.welch(x.reshape(1,x.size),fs,nperseg=256,
>>>                     nfft=1024,detrend=False,scaling='density')
>>> Pxx = Pxx.reshape(Pxx.size, 1)
>>> f = f.reshape(f.size, 1)
>>> fig2 = plt.figure(2)
>>> ax21 = fig2.add_subplot(211)
>>> ax21.plot(f, 10 * np.log10(Pxx), 'b')
>>> plt.xlabel('f [Hz]')
>>> plt.ylabel('Magnitude [dB]')
>>> plt.grid('on')
>>> plt.axis([0, 20, -40, 20])
```



```
>>> dt = 0.55
>>> pb = np.array([0,0.58-dt,0.58,1.27,1.27+dt,20])/(40)
>>> b = signal.remez(150, pb, [0, 1, 0], type='bandpass')
>>> w, h = signal.freqz(b)
>>> ax22 = fig2.add_subplot(212)
>>> ax22.plot(w / (2 * np.pi) * 40, 20 * np.log10(np.abs(h)), 'b')
>>> plt.xlabel('f [Hz]')
>>> plt.ylabel('Magnitude [dB]')
>>> plt.grid('on')
>>> plt.axis([0, 20, -40, 20])
```


Magnitude [dB]
f [Hz]
Magnitude [dB]
f [Hz]