Niko Krasowski
niko.krasowski@iwr.uni-heidelberg.de

Exercise 0

Deadline: None

Computer Setup

Windows

Download *Python(x,y)* via `http://code.google.com/p/pythonxy/wiki/Downloads` and install it. Make sure that – before installation – the installer does not complain about an existing Python version on your computer.

Although *Python(x,y)* comes already with a great variety of scientific *Python* packages, we might have to install additional dependencies:

For instance, we will make use of the *vigranumpy* package which you can download and install from this website: `http://www.lfd.uci.edu/~gohlke/pythonlibs/`

Now you can test whether your installation was successful: Find the Python(x,y) folder in the Windows start menu and launch the contained Spyder program. In the interactive interpreter on the bottom right, enter

```
import vigra
import numpy
import sklearn
```

If no output appears, your installation was successful. If the *sklearn* import failed, try to install the latest version from `http://www.lfd.uci.edu/~gohlke/pythonlibs/#scikit-learn`.

Ubuntu

Either compile everything yourself, use *pip*, or just execute the following line to install the system libraries we need:

```
sudo apt-get install build-essential python-dev python-numpy python-setuptools
    python-scipy libatlas-dev libatlas3-base python-matplotlib python-vigra python-
    sklearn spyder
```

If you want to start GUI programs such as `spyder`, always start them from the terminal; this way they inherit all necessary environment variables that have been set in `~/.bashrc`.

Mac

Install Homebrew (`http://brew.sh`) by running this command:

```
ruby -e "$(curl -fsSL https://raw.github.com/mxcl/homebrew/go)"
```

If you get errors here, follow these instructions:
`http://www.moncefbelyamani.com/how-to-install-xcode-homebrew-git-rvm-ruby-on-mac/`

Then, insert the Homebrew directory at the top of your `PATH` variable by adding the following line at the bottom of your `~/.bash_profile` file:

```
export PATH=/usr/local/bin:$PATH
```

Now open a new terminal and run the following command:

```
brew doctor
```

As long as you do not get `Your system is ready to brew`, try to fix the warnings.

In the next step, you can install Python using Homebrew via

```
brew install python
```

You can then install most python packages using *pip*:

```
pip install numpy
pip install scikit-learn
pip install matplotlib
pip install scipy
pip install scikit-image
pip install spyder
```

Note that it is not possible to install *vigranumpy* like that. In case you need this dependency for your project, we can assist you installing it.

Troubleshooting:

- If `pip install scipy` fails, you might have to run `brew install gfortran` first.
- If `pip install matplotlib` fails, try to do the following

  ```
  brew install freetype
  brew install libpng
  brew link --force freetype
  ```

  and then reinstall `scipy`.

The Scientific Python Ecosystem

In this course, we are going to use the *Python* programming language. There exist many highly usable and well maintained scientific libraries for Python. Packages that will be useful for us are:

- *numpy* (`numpy.org`), provides multi dimensional arrays and fast numeric routines that work with these arrays.
- *scipy* (`scipy.org`), a collection of many scientific algorithms for areas such as optimization, linear algebra, integration, interpolation, FFT, signal and image processing. Makes use of numpy arrays.
- *matplotlib* (`matplotlib.org`), a plotting library which provides a MATLAB like interface. Check out the great gallery with many examples.
- *scikit-learn* (`scikit-learn.org`), a quickly growing collection of machine learning algorithms, such as Support Vector Machines, Decision Trees, Nearest Neighbor Methods and many more. Their website offers good overview documentation and great examples, too.
- *scikit-image* (`scikit-image.org`), a collection of image processing algorithms.
- *vigranumpy* (`hci.iwr.uni-heidelberg.de/vigra`, a C++ library for multidimensional arrays, image processing and machine learning, developed in our group. It exposes most functions to Python via the vigranumpy module.

All of these packages can be easily installed on recent Linux distributions, such as Ubuntu.

Coming from MATLAB?

Read `http://www.scipy.org/NumPy_for_Matlab_Users`. Indexing starts at zero in numpy!

IPython

For an interactive python shell, start `ipython` within a shell. It supports auto-completion using the tab key, as well as showing the documentation of functions by appending a `?` to the function name.

Spyder

Spyder (`http://code.google.com/p/spyderlib`) is a MATLAB-like IDE for scientific Python. You can use the editor on the left or the interactive interpreter on the bottom right, just like in MATLAB.

Python

For a tutorial introduction to Python, we recommend `http://docs.python.org/tutorial`, sections 1 through 6 as well as 7.2 and 9.3. The following sub-sections, which are aimed at advanced users, can be skipped: 2.2, 4.7, 5.1.3, 5.1.4, 5.6, 5.7, 5.8, 6.4.

In the following, we give a quick overview of the language in order to get you started quickly.

Indentation

Python uses *indentation* to group statements as opposed to curly braces in C-like languages. Always indent using 4 spaces. Also note that braces around statements following `if`, `for`, etc. are omitted; instead, a colon : at the end is used.

```
if a == 42:
    for i in range(5):
        print i
else:
    b = 5
```

Variables

Python, as a scripting language, uses dynamic typing.

```
a = 42                    #int
b1, b2 = 42.0, float(42)  #float, cast to float
c = MyClass(a)            #instance of MyClass
d = None                  #special 'none' type
e = "Hello"               #string
f = [1,2,3]               #a list of values (mutable), f[0]=1
g = (1,2,3)               #a tuple of values (immutable), g[0]=1
h = {"x": 1, "y": 2}      #a dictionary h['x']=1, h['y']=2
```

Functions

Functions are declared with `def`. They take a list of required arguments and optional keyword arguments (with default values), and may `return` values.

```
def some_func(arg1, arg2, kwarg1=True, kwarg2=42):
    return arg1 + arg2
```

Control Flow

if, elif, else:

```
if (a or b) and c:
    print "x"
elif b:
    print "y"
else:
    print "z"
```

A for loop:

```
for i in range(0,10): # range [0,10)
    print i
    if i < 2:
        continue
    if i >= 7:
        break
```

Exercise: Fizz-Buzz (`http://en.wikipedia.org/wiki/Fizz_buzz`)

Write a program that prints the numbers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".

— http://imranontech.com/2007/01/24/using-fizzbuzz-to-find-developers-who-grok-coding

Lists, tuples and slicing

List, tuple or strings can be subscripted using *slice notation*.

```
s = "Hello"
print s[1:]
print s[2:4]
print s[0:-1:2]
print s[::2]
```

Exercise: Can you explain the output?

Modules

Additional functionality can be accessed by importing *modules*. If you want to import only parts of a module, use the `from module import symbol` notation, where the function can be optionally renamed using `as`:

```
import math
from math import pi as circle_number
print "cos(pi)=%f" % (math.cos(circle_number),)
```

Every python file you write can be imported from you in other files: If file `file1.py` defines a function `function1`, you can use it from `file2.py` in the same directory via `from file1 import function1`.

Files

Write `import` statements always at the beginning of your python file.
Always put your main function in `if __name__ == "__main__"` at the end of the file. This way, the main code won't be executed if you import this file from another file.

Numpy

A good tutorial for numpy can be found at `http://www.scipy.org/Tentative_NumPy_Tutorial`.

Import the numpy module via `import numpy`

Creating and calculating with arrays

```
import numpy
#create a 2D array of shape (4,6), data type is unsigned char
a = numpy.zeros((4,6), dtype=numpy.uint8)
print a.ndim  #2-dimensional
print a.shape, a.shape[0], a.shape[1]
print a.dtype #numpy.uint8

#uniform random numbers in [0,1)
b = numpy.random.random(a.shape)

#basic algebra for arrays
c = a+b
d = a*b
e = a/(b+1)
f = numpy.sqrt(b)
```

Array slicing

Slicing works similar to the slicing of lists, tuples and strings as introduced above. For multi-dimensional arrays, a slicing for each dimension is given (comma separated). Please see `http://docs.scipy.org/doc/numpy/reference/arrays.indexing.html` for a comprehensive overview.

Niko Krasowski
niko.krasowski@iwr.uni-heidelberg.de

```
a[:] = 1 #write a 1 to every entry
a[1,2] = 2 #assign 2 to the row 1, col 2 (indices start at 0!)

s = numpy.sum(a) #sum over all array elements
assert s == 7

a[:,0] = 42 #assign 42 to the 0-th column
            # ':' ==> select all rows
            # '0' ==> select the 0-th column

a[0,...] = 42 # fill 0-th element of first array dimension (here: column)
              # with 42's, no matter how many dimensions a has

b = a[:,0:2] #make a subarray: select only columns 0 and 1
             # ( range [0..2) )
```

numpy.where

To find the indices in an array where a certain condition matches, use `numpy.where`. The returned tuple of indices can be passed as an argument to the `[]` operator of any array:

```
import numpy
a = numpy.asarray([[2,5,4,3,1],[1,2,1,5,7]])
print a.shape
print a
print numpy.where(a == 2)
print numpy.where(a == 4)
print a[numpy.where(a == 1)]
```

Vigra

Import the *vigranumpy* module using `import vigra`.

The documentation can be found at `http://hci.iwr.uni-heidelberg.de/vigra/doc/vigranumpy/index.html`.

vigranumpy may be used, among many other functions, to read in images as `numpy` arrays or write them out to file[1]:

```
import vigra
img = vigra.impex.readImage('./myImage.png')
img.shape
# red channel
img = img[...,0]
img.shape
vigra.impex.writeImage(img, './redChannel.png')
```

Note that many functions expect the array to be of a certain data type. If you get errors like

```
Boost.Python.ArgumentError: Python argument types in
    vigra.filters.gaussianSmoothing(numpy.ndarray, int)
did not match C++ signature:
    gaussianSmoothing(vigra::NumpyArray<4u, vigra::Multiband<float>, vigra::StridedArrayTag> array,
                      boost::python::api::object sigma, vigra::NumpyArray<4u, vigra::Multiband<float>, vigra::StridedArrayTag> out=None,
                      boost::python::api::object sigma_d=0.0, boost::python::api::object step_size=1.0, double window_size=0.0,
                      boost::python::api::object roi=None)
    gaussianSmoothing(vigra::NumpyArray<3u, vigra::Multiband<float>, vigra::StridedArrayTag> array, boost::python::api::object sigma,
                      vigra::NumpyArray<3u, vigra::Multiband<float>, vigra::StridedArrayTag> out=None, boost::python::api::object sigma_d=0.0,
                      boost::python::api::object step_size=1.0, double window_size=0.0, boost::python::api::object roi=None)
```

you have forgotten to cast to the correct data type (note that it searches above for a `vigra::NumpyArray<4u, vigra::Multiband<float>, vigra::StridedArrayTag>`, which has pixel type float). In this case, you can cast to 32-bit float using `array.astype(numpy.float32)`. Other vigra functions may use data type `numpy.uint8` or `numpy.uint32`. Additionally, check that your array has the correct shape, in this case, the function expects an array with `ndim == 3` or `ndim == 4`.

[1]If you did not manage to install *vigra* on your machine, you might want to check out *scikit-image* for some of those operations.

Matplotlib

The recommended way to import matplotlib is:

```
import matplotlib; matplotlib.use("Qt4Agg")
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import pyplot as plot
```

See `http://matplotlib.org/gallery.html` to copy/paste some code.

Example: read and display image.

```
import numpy, vigra
import matplotlib; matplotlib.use('Qt4Agg')
from matplotlib import pyplot as plot

img = vigra.impex.readImage("myImage.png")[...,0] # read in red channel
plot.figure()
plot.gray() #select grayscale color map
plot.imshow(img)
plot.show()
```

Scikit-learn

Scikit-learn is a very powerful machine learning toolbox. Most of the algorithms contained in this package will be reviewed in the Pattern Recognition lecture. A tutorial for *scikit-learn* including detailed descriptions of the algorithms is available at `http://scikit-learn.org/stable/tutorial/index.html`.

Exercises (optional, no pts)

1. Plot $sin(x)$ for $x \in [-1, 1]$. Label the axes and add a title. Then save the figure as *png*.

   Implementation hints:
   Use `numpy` and `matplotlib.pyplot`.

2. Write a function computing $n!$ and call the function from your *main* function for $n = 5$.

3. Find and output the eigenvalues and eigenvectors of the following matrix:

$$A = \begin{pmatrix} 4 & 0 & -1 \\ 2 & 5 & 4 \\ 0 & 0 & 5 \end{pmatrix} \tag{1}$$

   Implementation hints:
   Check out the documentation page of `numpy.linalg` for eigenvalue computation.

4. Concatenate the eigenvectors as columns in a matrix P, compute P^{-1}, and confirm that $P \cdot \Lambda \cdot P^{-1} == A$, where Λ is the diagonal matrix of the eigenvalues of A. What does `a * p` do?

   Implementation hints:
   In `numpy`, find a function for matrix multiplication and for creating a diagonal matrix. Matrix inversion is implemented in the `numpy.linalg` package. You might also want to look at `numpy.all`.