

Version: January 21, 2020

© Computer Vision Laboratory

Linköping University

Image Processing in PYTHON

Tutorial for TSBB15

1 Introduction

During this exercise, the goal is to become familiar with Python and the NumPy library. You should also get a better feeling for how images are represented as matrices as well as the connection between mathematical expressions and the Python code to implement them.

1.1 Matlab vs Python

As many other courses use Matlab we will sometimes compare how something is done in python with how it is done in Matlab. If you are unfamiliar with Matlab, you can ignore these parts. Unlike Matlab, Python is a general programming language used for many things, such as web servers and games aside from the more mathematical subjects covered in this course. In practice this means slightly more typing to specify what you want to do, it but also allows you to structure programs more easily than Matlab. For example, in Python the numerical operations are not included in the language, rather they are implemented as a library called NumPy that must be imported explicitly. This library is written in C/C++ and is highly optimized. Just like in Matlab one can of course perform explicit for-loops over data, but executing such a loop will likely take more time than figuring out how to call the C++ implementations.

1.2 Preparations

Before the exercise you should have read trough this document. If you are not familiar with Python syntax in general, you should look try to look into this more thoroughly than can be covered in this guide.

The first line includes the OpenCV module, which is a library of computer vision related operations that is used in this course. The second line activates the parts specific to this course:

```
$ module load prog/opencv/3.4.3
$ module load courses/TSBB15
```

Start the standard Python interpreter by typing:

```
$ python3
```

or for the alternative Ipython interpreter:

```
$ ipython3
```

Note that the commands `python` and `ipython` (without the 3 at the end) will also work, but will instead start an older version of the python language, without the course-packages installed

Starting the python interpreter should print something like:

```
→ python-tutorial python
Python 3.6.6 |Anaconda custom (64-bit)| (default, Jun 28 2018, 17:14:51)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Figure 1: The standard python interpreter

While for Ipython the prompt should look like this:

```
(base) → ~ ipython
Python 3.7.3 (default, Mar 27 2019, 22:11:17)
Type 'copyright', 'credits' or 'license' for more information
IPython 7.6.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]: █
```

Figure 2: The Ipython interpreter

In general, Ipython is better for interactive use as it has some additional user-friendliness compared to the standard interpreter. Python files are text files with the ending `.py`, and can be written in any text editor. To run such a file, type: `python3 thefile.py` in the terminal.

If you have issues with these initial steps, or other problems with the lab-computers in general send an email to `helpdesk@liu.se` with a description of your problem.

2 Getting started with Python

The first thing we need to do after having started a Python interpreter is to import some libraries that we will use, write this in a terminal, or at the beginning of a `.py` file:

```
import numpy as np
import scipy
from matplotlib import pyplot as plt
```

The first line imports the NumPy library, and renames it to the more compact `np`. This is useful as we will often call functions from NumPy, and all such calls will be prefixed with `np`, for example, `np.power(x,2)` corresponds to: x^2 element wise, when x is a NumPy array. The built-in Python function `pow` will also work, but `np.power` is more explicit, and therefore better.¹

The second line imports SciPy while maintaining its original name, the final line imports the sub-module `pyplot` from the module `Matplotlib` and giving it the name `plt`, again to make it easy to call often.

To see what a module can do, you can call the built-in help function on the imported name, for example:

```
help(np)
```

This will open a text browser with a description of all the functions and objects in the `np` namespace. In practice it is often more convenient to look up the documentation on the web, for NumPy/SciPy it can be found at: <https://docs.scipy.org/doc/>. This documentation is generally up-to date and very well written. Exit the text browser by pressing `q`, this should bring you back to the Python prompt.

¹<https://www.python.org/dev/peps/pep-0020/>

3 Simple array operations, indexing and visualization

A NumPy array, more or less equivalent to a matrix in Matlab, is created by calling the constructor for the array object like:

```
row_vector = np.array([[1,2,3]], dtype='float32')
small_matrix = np.array([[1,2,3],[4,5,6],[7,8,9]], dtype='float64')
column_vector = np.array([[1],[2],[3]])
black_image = np.zeros((64,128,3))

#Note that the matrix operations will behave strangely on this one,
#compared to the row and column vectors respectively.
#It gets treated as a row or column vector depending on the operation.
a_one_dimensional_array_that_is_not_a_vector_except_sometimes = \
    np.array([1,2,3,4])
four_element_row_vector = \
    np.reshape(a_one_dimensional_array_that_is_not_a_vector, (-1, 1))
four_element_column_vector = \
    np.reshape(a_one_dimensional_array_that_is_not_a_vector, (1, -1))
```

This creates two vectors, one matrix and a 3-dimensional array containing zeros. In practice the one dimensional arrays can be treated as linear algebra vectors, the two dimensional array as a matrix. The higher dimensional arrays are a kind of generalization of a matrix to have more than two indices. Such objects are sometimes called Tensors, but this is not strictly correct mathematically and not really used in this course. Instead we will call a something a Tensor if it is a tensor, and an array if it is an "indexable lump of numbers".

Some simple operations that can be performed on the objects created above are:

```
# Scalar product (vector * vector = scalar):
scalar = row_vector @ column_vector
print("Scalar: {}".format(scalar))

# Outer product (vector * vector = matrix):
matrix = column_vector @ row_vector
print("Matrix:\n {}".format(matrix))

# matrix * vector product:
v1 = small_matrix @ column_vector
print("matrix*vector:\n {}".format(v1))

# vector * matrix product:
v2 = row_vector @ small_matrix
print("vector*matrix:\n {}".format(v2))

# Elementwise multiplication (same as .* in Matlab)
em = matrix * small_matrix
print("Element wise:\n {}".format(em))
```

Here the @ symbol denotes matrix multiplication, and the * symbol denotes element-wise multiplication.

As we are working with images it is sometimes useful to visualize them, this can be done using Matplotlib as:

```
image = plt.imread('/courses/TSBB15/images/capybaras.jpg')
plt.imshow(image)
plt.show()
```

The first line reads the image 'copybaras.jpg' from the hard drive. The second line enqueues an image for display at some point in the future. The final lines blocks the interpreter to show the image. There should be a magnifying glass icon that can be used to zoom in on the image. Use this zoom in and pan around in the image with the arrow icons. This can be very useful when looking at details in intermediate steps of your calculations later. Close the window to continue/unblock the interpreter. (If using ipython the interpreter might not be blocked)

As this image is larger than we need, it is a good idea to make it smaller. This can be done in multiple ways, but a kind of simple first approach is to only keep one of five pixels. This can be done using the fancy indexing in numpy. For example:

```
smaller_image = image[0:-1:5, 0:-1:5, :]
plt.imshow(smaller_image)
plt.show()
print("Original image size is: {}".format(image.shape[0:2]))
print("New image size is: {}".format(image.shape[0:2]))
```

The first line here uses fancy indexing to pick out only every fifth pixel. It starts at the first pixel (index 0), and stops at the last one (index -1), with the step between pixels being 5. This is done for the first two dimensions (the height and width dimensions). For the final dimension (the color dimension) all information is kept. The final two lines print the sizes of the original image, and of the new smaller image.

Note that this is a fairly terrible way to downsample images, as it introduces aliasing effects. The goal is simply to illustrate simple indexing operations.

3.1 List comprehension syntax and timing

An important difference in Python compared to other languages is that code blocks are defined using whitespace and indentation, where C++ instead uses curly braces. This means that correct indentation is important, and not just something that will make others angry when you get wrong. If you have problems with this, set whatever text editor you are using to show whitespace characters. Try to stick with using 4 blank spaces as one indentation level.

Python syntax has some features that makes it possible to, for example iterate over a list of objects:

```
things_list = ['a', 'b', 1, 2, 3, 4, 1.2, 1.3, 1.4]
numbers_list = [item for item in things_list if isinstance(item, (int, float))]
letters_list = [item for item in things_list if isinstance(item, str)]
print("=====")
for number in numbers_list:
    print(number)
print("=====")
for idx, letter in enumerate(letters_list):
    print("Letter number: {}, is {}".format(idx, letter))
```

The part in square brackets is a list-comprehension, when it is possible to express something as one it is typically easier to read, and runs much faster than a traditional for-loop. This can be verified easily when using ipython, by typing:

```
%timeit somefunction
```

Ipython has a few "magic" commands that are prefixed by the % sign. When using the standard interpreter you need to write your own timing function. Since the timeit command only works for a single statement you will need to wrap the code to benchmark in a function.

```
def loop_function(N):
    """ Build a list using a loop """
    l = []
    for n in range(N):
```

```

        l.append(n)
    return l

def listcomprehension_function(N):
    """ Build a list using a list comprehension """
    return [n for n in range(0,N)]

%timeit loop_function(100)
%timeit listcomprehension_function(100)

```

Finally, do read the documentation for functions you are not sure of what they do, the numpy documentation is comprehensive and up to date. If you are not sure of what a function does after reading the documentation, call it with some arguments and see what falls out. Perhaps most importantly, even if you are mostly sure of what a function does, *verify it anyway* by calling it with some arguments that should produce a simple output. **Just because a command runs without crashing it does not mean that it did what you want.**

3.2 Many ways to read an image

Reading an image can be done using pillow, scikit-image, opencv or matplotlib

```

image_filename = 'someimagefile'

# using pillow
from PIL import Image
im = np.array(Image.open(image_filename))
print(im.shape)

# Using opencv
import cv2
im = cv2.imread(image_filename)
print(im.shape)

# using scikit-image
import skimage.io as skio
im = skio.imread(image_filename)
print(im.shape)

# using matplotlib
import matplotlib.image as mpimage
im = mpimage.imread(image_filename)
print(im.shape)

plt.imshow(im)
plt.show()

```

Note that depending on what you use to read the image, the image size will be different, as the libraries do not treat the alpha channel the same way. Also note that OpenCV by default permutes the color channels to the order BGR, instead of the usual RGB.

4 Exercises

4.1 Plotting

Begin by creating a sine function:

```
sine_curve = np.sin((np.pi*2) * np.linspace(0,1,10000))
plt.plot(sine_curve)
plt.show()
```

The plot function can take a large number of different parameters. Look them up at: http://matplotlib.org/api/plot_api.html.

There is also a lot of example plots with the code to generate them at: <https://matplotlib.org>
Question: How would you make a plot with a magenta dash-dotted line with hexagrams at each data point?

4.2 Slicing and advanced indexing

Indexing a numpy array is done using the `arr[idx]` syntax, for a multidimensional array we use `arr[idx1, idx2, idx3, ...]`. Writing `a = arr[idx]` assigns a to the value at `idx`, while `arr[idx] = a` assigns the position `idx` to value `a`. We can however do more advanced indexing, such as:

```
arr = np.zeros((512, 512, 3))
arr[:, :] = np.array([0, 0, 1])
arr[200:, :] = np.array([1, 1, 1])

# These arrays have the same size as arr
# x contains the x coordinate of each position, y contains y
(x, y) = np.meshgrid(np.arange(512), np.arange(512))

x_0 = 60
y_0 = 100
c_x = (x - x_0)**2
c_y = (y - y_0)**2
inside = np.sqrt(c_x + c_y) < 30

# assign a color to the selected pixels
arr[inside, :] = np.array([1, 1, 0])

plt.imshow(arr)
plt.show()
```

Question: What does the first three lines do?

Now add a snowman (three stacked circles) to the image using the same indexing operations demonstrated above.

4.3 Test-pattern generation

We start by generating two simple gradients (note that `arange` does not include its endpoint, thus the seemingly asymmetric range):

```
(x,y) = np.meshgrid(np.arange(-128,129,1),np.arange(-128,129,1))

plt.figure(1)
plt.imshow(x)
plt.figure(2)
plt.imshow(y)
```

Now, use these arrays to generate an image of the function $r = \sqrt{x^2 + y^2}$.

Question: Write down the commands you use here:

Question: What is the function `r`?

We will now use this function to generate our first test image:

Generate the function $p_1 = \cos(r/2)$ and display it.

Generate the Fourier transform of this pattern as variable, `P1`, using `np.fft.fft2`, and plot it.

Take the absolute value of the Fourier transform using `np.abs`.

Question: What does the Fourier transform look like? Explain why:

Question: What is the difference between `np.fft.fftshift` and `np.fft.ifftshift`

We will now create a test pattern with a slightly more interesting frequency content. Use the variable `r` to create the function $p_2 = \cos(r^2/200)$.

Plot the image `p2`, and its Fourier transform `P2`.

Question: The function $p_2 = \cos(r^2/200)$ appears to be rotationally symmetric, and its Fourier transform should then be rotational symmetric. Why then isn't the Fourier transform of `p2` rotationally symmetric?

To get a slightly better result, we now cut off the outermost periods of the `cos()` signal:

```
p2 = np.cos((r**2 / 200.0)) * ( (r**2 / 200.0) < (22.5 * np.pi) )
P2 = np.fft.fft2(p2)
*** is the exponentiation operation, so x**2 = x^2, x**3 = x^3
```

You should now get a somewhat more correct representation of the transform P2, but note that the superimposed ringings are still there.

To generate a test image with a controlled frequency content is actually not that easy. The main problem is to create a test image which locally contains one frequency and has a nice behavior in the transition from the dc part to the low frequency part.

4.4 Filtering

We will now low-pass filter our test image. We start with filtering in the Fourier domain. Here the following variables will probably come handy:

```
u=x/256*2*np.pi
v=y/256*2*np.pi
```

Question: How do you generate a low-pass filter in the Fourier domain?

Assuming that the Fourier domain spans the range $[-\pi, \pi]$, use the method you describe above to generate an ideal LP-filter with the cut-off frequency at $\rho = \pi/4$ (Hint: A scaled r , and logical expressions are useful here.).

Use your filter on the transform P2 of your test image p2.

Question: How is filtering in the Fourier domain computed?

Compute the inverse Fourier transform of the result from your filtering, and compare this with the original image p2.

Question: Since the image p2 contains radially increasing frequency, one could expect that the result was attenuated outside a certain radius, and largely unaffected in the image centre. Why didn't this happen here?

We will now apply a box filter in the spatial domain. Instead of multiplication, the filtering now consists of a convolution:

```
#import convolution function
from scipy.signal import convolve2d as conv2

lp = np.ones((1,9), dtype='float32')/9.0
p2fs = conv2(lp, np.transpose(lp))
#Transpose can also be called like:
#lp.T
```

Question: This is not good either. Why does this filter behave differently in different parts of the image?

Now, try a Gaussian filter instead:

```
sigma = 3.0
lp = np.atleast_2d(np.exp(-0.5 * np.square(np.arange(-6,7,1)/sigma)))
lp = lp / np.sum(lp)
```

Question: Why does this filter work better? (Hint: Look at the filter `lp*lp`.)

Question: Why is the filter divided with its sum?

4.5 Multi-dimensional arrays

We will now read a color image from disk:

```
image = mpimage.imread('some_image_file.png').astype('float32') / 255
```

The `astype('float32')` part converts the image to float, and the division with 255 makes sure the range is $[0, 1]$ rather than $[0, 255]$.

Check the size of the array containing the image

```
print(image.shape)
```

```
plt.imshow(image)
plt.imshow(image[:, :, 1])
```

Question: Which component is which?

Convert the image to grayscale using the formula $gray = 0.299R + 0.587G + 0.114B$, and plot it where the color image now is. Compare the grey-scale image with the RGB components. (This is best done in a function)

4.6 Complex valued images

We will now combine the results of two Gaussian derivative filterings into one complex-valued field describing image orientation. Computing the derivative of a discrete signal is in general impossible, since

it is the limit of a difference quotient, and we only have a sampled signal. It is however possible to compute a *regularised derivative*, i.e. the derivative convolved with a smoothing kernel:

$$\frac{\partial}{\partial x}(f * g(\sigma))(x) = (f * \frac{\partial}{\partial x}g(\sigma))(x) = (f * \frac{-x}{\sigma^2}g(\sigma))(x)$$

Where $g = \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{x^2}{2\sigma^2}}$. Thus, the regularised 1D derivative filter becomes:

```
df = np.atleast_2d(-1.0/np.square(sigma) * np.arange(-6,7,1) * lp)
```

Using `lp` and `df`, we can compute regularised partial derivatives along the x and y axes. Apply them to the test pattern `p2` (see section 4.3), you can also try the image, `image`, that you loaded before. Now we store the results of filtering with these kernels as a complex field, and visualise the result using the `gopimage` command:

```
dx = conv2(conv2(p2,df,mode='same'),lp.T,mode='same')
dy = conv2(conv2(p2,lp,mode='same'),df.T,mode='same')
z = dx + 1i*dy
```

```
from lab2 import gopimage
```

```
gopimage(z)
```

Each complex value in `z` can now be seen as a vector $(\text{Re}(z) \ \text{Im}(z))^T$ that approximates the image gradient.

Use the zoom tool to locate pixels of different colors, then extract the corresponding complex values from `z`. For instance you could try:

```
print(z[139,92])
```

Question: Describe what the colors mean:

Now, double the argument of the gradient image:

```
z2 = np.abs(z) * np.exp(1i * 2 * np.angle(z))
gopimage(z2)
```

Question: Compare the results, and explain why the double-angle representation is useful.

Have a look at the gradient magnitude, by computing the absolute value of the `z` variable.

Question: Why are there periodic variations in the magnitude?

4.7 2D tensor images

If we want to estimate the local orientation, the periodic variation in magnitude we saw above is not desirable. One way to remove the periodic variations is to average the double angle representation `z2`.

Why is it not possible to average \mathbf{z} ? Another way to avoid this problem is to replace the complex valued image \mathbf{z} with a tensor image \mathbf{T} . Each position in the tensor image \mathbf{T} contains the outer product of the vector $\mathbf{f} = (f_x \ f_y)^T$, i.e. $\mathbf{T}(\mathbf{x}) = \mathbf{f}(\mathbf{x})\mathbf{f}(\mathbf{x})^T$. If we perform averaging in the tensor representation, we can smear out the magnitude such that it is largely independent of the local signal phase.

$$\mathbf{T} = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \begin{pmatrix} f_x & f_y \end{pmatrix} = \begin{pmatrix} f_x f_x & f_x f_y \\ f_y f_x & f_y f_y \end{pmatrix} \quad \text{and} \quad \mathbf{T}_{lp}(\mathbf{x}) = (\mathbf{T} * g_x * g_y)(\mathbf{x})$$

Since \mathbf{T} is symmetric, we only need to store three of the components in NumPy.

Now generate the tensor.

```
T = zeros((256, 256, 3))
T[:, :, 0] = dx * dx
T[:, :, 1] = dx * dy
T[:, :, 2] = dy * dy
```

In order to be sure we implemented this correctly, it is advised to look at the produced tensor values. While calling `imshow` might work, it will not behave correctly since the tensor is not an RGB image. It is better to look at each channel separately as grayscale images.

Next we apply Gaussian smoothing with the appropriate standard deviation σ on all the tensor components. Try to find a σ -value that gives just enough smoothing to remove the magnitude variation.

```
sigma = 3.0
lp = np.atleast_2d(np.exp(-0.5 * (np.arange(-10, 11, 1)/sigma)**2))
lp = lp / np.sum(lp) #normalize the filter

Tlp[:, :, 0] = conv2(conv2(T[:, :, 0], lp, mode='same'), lp.T, mode='same')
Tlp[:, :, 1] = conv2(conv2(T[:, :, 1], lp, mode='same'), lp.T, mode='same')
Tlp[:, :, 2] = conv2(conv2(T[:, :, 2], lp, mode='same'), lp.T, mode='same')
```

Question: Write down the required `sigma` value:

Question: The actual local orientation property is an angle modulo π . How is this angle represented in the tensor?

4.8 Images as vectors

This last exercise emphasizes the fact that we can treat an image as a vector instead of a multidimensional matrix. Treating an image as a vector has the advantage that we only need to keep track of one index instead of several. Load the file `/courses/TSBB15/mystery_vector.npy` to obtain an image stored as a vector. The exercise is to find the maximum value and the corresponding position of this value given in image coordinates. Finally reshape the vector to the original image. The original image consists of 320 columns and 240 rows.

HINT: One useful command is `unravel_index`.

Question: What is the position of the maximum value in image coordinates?

5 Leftovers

This section covers some things that did not fit in the rest of the guide, such as unit tests, development tools and Jupyter notebooks. None of this is strictly part of the course, but they are still useful to know about as they reduce the difficulty of finding bugs.

5.1 Unit tests

When writing programs larger than a few lines it is often useful to test the intermediate steps of the program in isolation. There are (at least) two reasons for this: It helps you define what each part of the program should actually do, and it makes sure that it actually does what you intended in isolation from the other parts. For larger projects involving multiple people unit tests can allow you to test each component in isolation for some simple cases.

A unit-test is in practice a nothing more than a function that calls the function you want to test with arguments that should produce some known output, for example:

```
def test_addition():
    a = 10
    b = 15

    c = 25

    assert a + b == c
```

This function tests that the addition operation behaves as expected. Often you would write multiple tests so that you can be sure that edge cases are handled correctly. A more complicated example for a function that projects a point in an image:

```
def project_point_in_image(point, camera_matrix):
    image_point = camera_matrix @ point

    return image_point
```

With the tests being:

```
def project_camera_position():
    #Project a point that is at the same position as the camera into the image
    pass

def project_behind_camera():
    #Project a point that is behind the camera into the image
    pass

def project_point():
    #Project a sensible point that is in front of the camera into the image
    pass
```

Try to implement the missing test cases. It is recommended to use the same camera matrix in all tests, but different points. Note that the point of this exercise is as much to implement the tests as it is to think about what the result should be in each case.

5.2 Jupyter notebooks

Jupyter notebooks let you execute python in an environment similar to Matlab scripts, where you can freely mix text, code and figures. It is typically used for prototyping things rather than full-on programming as they get complicated with large amounts of code. To start the notebook server type "jupyter notebook" in a terminal. The actual notebook is opened as a web page. Going into more detail about this is out-of-scope for this tutorial, but it is useful to know that it exists.

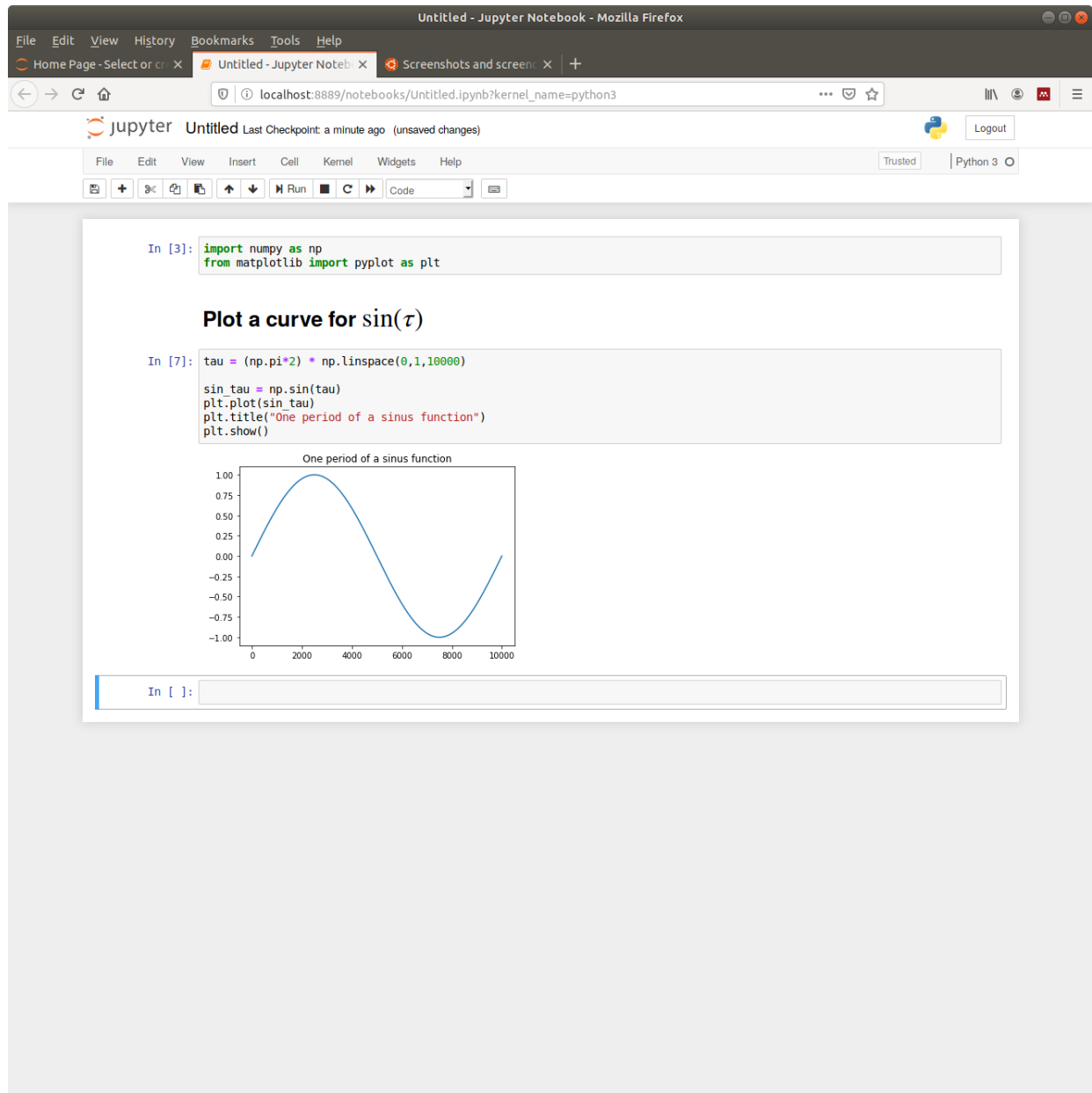


Figure 3: An example jupyter notebook.

5.3 Development Tools

There exist a large number of different development tools for python, ranging from a text-editor like sublime text that provides minimal help, but also makes it easy to get started, to full IDEs, such as pycharm. Getting started with an IDE might take some setup time, and require a powerful computer but will provide auto complete functionality and integrated debugging.