Computational Vision (Psy 5036): Python

So far in this course we've tried to emphasize concepts usually with simple examples. We'll now go over tools that can be applied to state-of-the-art problems in computational vision. Mathematica is excellent for learning concepts, and for many high-end applications. However, it is not so good for specialized work in the field where most others are using different tools. Matlab and increasingly Python have large user communities who are building tools that we can build on. That's the goal. But first an introduction to Python as an environment for scientific programming.

Scientific programming with Python relies on several key add-on packages (see Installation cell below). Matlab-like functionality is provided by the NumPy, and other packages. Mathematica-like notebook functionality is provided by Jupyter, an open-source web application. IPython is an interactive environment, that includes notebook functionality, and provides a kernel for Jupyter. Jupyter supports other languges (R, Julia,..), and is now replacing IPython for notebooks.

The purpose of this notebook is to provide a very brief introduction to scientific Python with a guide to resources for learning more.

Rationale for scientific python and IPython/Jupyter notebooks

See this 2014 article in Nature (http://www.nature.com/news/interactive-notebooks-sharing-the-code-1.16261). For an introduction to IPython (http://ipython.org/notebook.html).

For scientific python using notebooks, see in-depth series (https://github.com/jrjohansson/scientific-python-lectures).

For a comparision of Python/NumPy with Matlab, see here (https://docs.scipy.org/doc/numpy-dev/user/numpy-for-matlab-users.html).

Starting IPython in the middle

In order to run this notebook on your computer, you will need to have Python and various packages installed. See the Installation cell below.

Find the directory where you've downloaded this notebook, and then from a terminal command line go the directory (or parent directory) and type: jupyter notebook. ('ipython notebook' should also work).

This command will bring up a browser window from where you should see this notebook listed in your browser window. You can load it from there.

Installation

The hardest, or at least the most frustrating, aspect of Python can be installation. There are a number of package managers. I recommend Anaconda (https://store.continuum.io/cshop/anaconda/). We'll be using Python 2.7.

In addition to Python and Juypter (http://jupyter.org/), you will need numpy (http://www.numpy.org/), matplotlib (http://matplotlib.org/) and the scipy library (http://www.scipy.org/scipylib/index.html). All of these are part of the scipy stack (scipy.org) for general purpose scientific programming.

For image processing, we'll use scikit-image (http://scikit-image.org/). This, like the machine learning module scikit-learn (http://scikit-learn.org/stable/index.html), is built on the scipy stack.

There is also increasing support for "Open Source Computer Vision OpenCV (http://opencv.org)" on python (http://docs.opencv.org/master/d6/d00/tutorial_py_root.html#gsc.tab=0).

For Bayesian computations using MCMC sampling see pymc (http://pymc-devs.github.io/pymc/).

Style

Let's first cover some style used in notebooks. Right off the bat, you can create various kinds of cells: Raw NBConvert, various headings, or a Markdown cell. Try double-clicking on some of the cells in this notebook. Code cells are for code.

This cell is Markdown. You can type in LaTeX commands and have them formatted. E.g. try putting the next line between double dollar signs:

p(y_i|x)= \frac{e^{y_i(w.x+b)}}{{1+e^{(w.x+b)}}}

After getting acquainted with the menu items and buttons of the IPython notebook interface, take a look at these notes on: IPython's Rich Display System (http://github.com/odewahn/ipynb-examples/blob/master/Part%205%20-%20Rich%20Display%20System.ipynb). Try copying in cell content in this notebook to try out displaying different kinds of content.

You have access to unix shell commands too. Try ls, !ls, and some of the magics: %ls, %lsmagic.

In [ ]:

Making and plotting lists

In [46]:
```
import numpy as np
```

To get started, let's look at some simple python coding examples. We need to load numpy to handle vectors and matrices. To make lists in Mathematica we typically used Table[], e.g.

```
Table[Sin[x], {x, 0, 1, .1}];
```

In python, we'll use "list comprehensions". Create a code cell and try these:

```
[sin(x) for x in arange(0,1,.1)]
```

```
[sin(x) for x in linspace(0,1,10)]
```

But wait! Python needs to know where these functions came from. arange(), linspace(), and sin() are all numpy functions. We imported numpy functions with the shorthand "np", so you will need to write:

```
np.sin(x), and np. arange(0,1,.1).
```

For more on creating and manipulating numerical lists in NumPy see scipy page (http://scipy-lectures.github.io/intro/numpy/index.html).

In [ ]:

Store the values in sl:

In [49]:
```
sl=[np.sin(x) for x in np.arange(0,10,.1)]
```

Let's plot the values in sl. To do this, we'll need to import matplotlib, and in particular the pyplot module, or plt for short. If you want a more matlab like plotting environment, you can use pylab:

```
from pylab import *
```

For more information on plotting, see scipy notes (https://scipy-lectures.github.io/intro/matplotlib/matplotlib.html#ipython-and-the-pylab-mode), and Lecture 4 in the scientific python notebook series (http://nbviewer.ipython.org/github/jrjohansson/scientific-python-lectures/blob/master/Lecture-4-Matplotlib.ipynb).

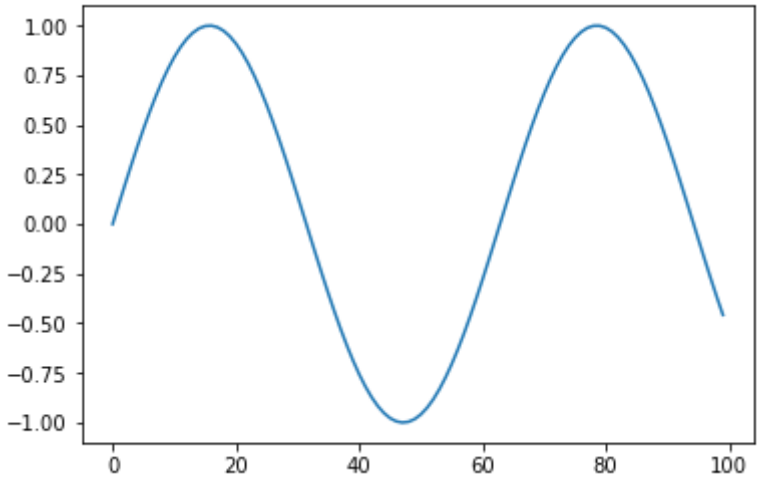
How to get information about a function? In Mathematica you can type ?Sin. In an IPythoncode cell try typing:? sin. Now try ?np.sin.

In [47]:
```
import matplotlib.pyplot as plt
```

With the magic function the next cell, your plot should appear inside the notebook. Otherwise, without the magic function, the plot appears in a separate window.

In [38]:
```
%matplotlib inline
```

In [39]:
```
plt.plot(sl)
plt.show()
```



In [ ]:
```
```

We made 2D lists in Mathematica using the Table[] function like this:

```
Table[Sin[x] Cos[y], {x, 0, 1, .1}, {y, 0, 1, .1}];
```

Now try using the list comprehension syntax for python:

```
[[sin(x)*cos(y) for x in arange(0,1,.1)] for y in arange(0,1,.1)] or

[[sin(x)*cos(y) for x in linspace(0,1,10)] for y in linspace(0,1,10)]
```

Again remember to specify the numpy class using np.

In [43]:
```
#If you remove the semicolon and it is the last line, the next line will
 print out the values
[[np.sin(x)*np.cos(y) for x in np.arange(0,1,.1)] for y in
np.arange(0,1,.1)];
```

In [45]:
```
#but it won't if you assign a variable name to the array

sl2=[[np.sin(x)*np.cos(y) for x in np.arange(0,1,.1)] for y in
np.arange(0,1,.1)]
```

Getting parts of vectors and arrays

Accessing array values or slicing (http://scipy-lectures.github.io/intro/numpy/numpy.html#indexing-and-slicing)

In [32]:
```
#make a matrix with 5 rows and 6 columns
sl3=np.array([[x+y*6 for x in np.arange(0,6,1)] for y in
np.arange(0,5,1)])
#what happens if you don't use np.array() in the above line?

print np.array(sl3),'\n\n(#rows, #columns)=',np.shape(sl3),"#
elements:",np.size(sl3)
```

```
[[ 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 26 27 28 29]]

(#rows, #columns)= (5, 6) # elements: 30
```

In [33]:
```
#note that unlike Mathematica and Matlab (but like C), indexing starts w
ith 0
#first row
sl3[0]
```

Out[33]:
```
array([0, 1, 2, 3, 4, 5])
```

In [34]:
```
#first column
sl3[:,0]
```

Out[34]:
```
array([ 0,  6, 12, 18, 24])
```

In [35]:
```
#Try accessing the element in the 3rd row, 4th column. It isn't sl3[3,4]
|

#Get the first 2 elements of sl. It isn't sl[0:1].
```

In [36]:
```
#Now get the submatrix 3rd and 4th row, 4th through 6th columns
sl3[2:4,3:6]
```

Out[36]:
```
array([[15, 16, 17],
       [21, 22, 23]])
```

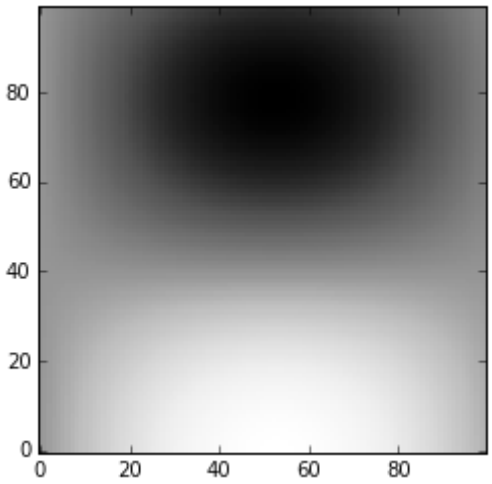
Defining functions

In [37]:
```
def f(x,y):
    return([[np.sin(3*x)*np.cos(4*y) for x in np.arange(0,1,.01)] for y
in np.arange(0,1,.01)])
```

In [38]:
```
#Visualize f(x,y) using imshow()

import matplotlib.cm as cm

n = 10
x = np.linspace(-3,3,4*n)
y = np.linspace(-3,3,3*n)
X, Y = np.meshgrid(x,y)
plt.imshow(f(X,Y),cmap = cm.Greys_r,origin='lower',interpolation='neares
t');
```



IPython Examples

For a directory of IPython notebook examples (https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks). E.g. here's a demo of receptive field models (http://nbviewer.ipython.org/github/jonasnick/ReceptiveFields/blob/master/receptiveFields.ipynb)

Python image processing

Look over the examples at the scikit-image site: http://scikit-image.org/docs/dev/auto_examples/ (http://scikit-image.org/docs/dev/auto_examples/)

In particular, take a look at the normalized cut method (http://scikit-image.org/docs/dev/auto_examples/plot_ncut.html) for producing segmentations, and related methods, sometimes referred to as "super-pixels" (http://scikit-image.org/docs/dev/auto_examples/plot_segmentations.html#example-plot-segmentations-py).

Contrast these methods with the edge-detection methods you have learned about.

Here are some demonstrations (http://gandalf.psych.umn.edu/users/kersten/kersten-lab/courses/Psy5036W2015/Lectures/17_PythonForVision/Demos/index.html) designed by Weichao Qiu for the chapter by Yuille and Kersten, relevant to this course, some of which are duplicated in individual links on the class web page.

Python computer vision demonstrations using OpenCV

In the next lecture we cover motion perception and optic flow. We will see there an example of using OpenCV to track feature points.

Python neural network resources

For python code, look at neurolab (https://pythonhosted.org/neurolab/index.html). This has several topics that should look familiar.

For python code to simulate spiking neurons, see http://briansimulator.org (http://briansimulator.org).

```
In [11]: ?np.arange
```

```
In [ ]:
```