NCAR SEA 2015: IPython Notebook Tutorial

R. Saravanan

Texas A&M University

sarava@tamu.edu

Notebook at http://nbviewer.ipython.org/urls/dl.dropboxusercontent.com/u/72208800/docs/NCAR-SEA15.ipynb

PDF slides at https://dl.dropboxusercontent.com/u/72208800/docs/NCAR-SEA-IPython-tutorial.pdf

---

Mindmeldr classroom interaction software

Need a Facebook or Twitter account (do not use Google). You can also create a trial account.

Web-site: http://mindmeldr.com

Join group `t-ncarsea15` to follow along and interact

Instructions: http://mindmeldr.com/_static/docs/html/start_student.html

---

What is your Python background?

A. Advanced

B. Beginner

C. None

What is your IPython Notebook knowledge level?

A. Advanced

B. Beginner

C. None

Do you have Python installed and working on your laptop?

A. Yes, on OS X

B. Yes, on Windows

C. Yes, on Linux

D. No

What would you like to learn from this tutorial?

Alternatives for hands-on exercises

1. Use your own Python installation on Windows/Mac/Linux
2. Install Anaconda Python distribution on your laptop from http://continuum.io/downloads
3. Use cloud server created specifically for this workshop. (*Note*: All files will be deleted at the end of the today's session.)

Cloud server

Open http://sea.gterm.net in your browser

Select a lowercase username (say `jsmith` for `Joe Smith`)

Enter group code `0c6c-c5cd-556a-197b` and click *Authenticate*

This will create your account. Copy the generated passcode and save it somewhere, or email it to yourself.

Starting the notebook on the cloud server

Create a new session

Type the following command:

```
gnbserver
```

This will print out a bunch of lines, one of which contains a port number starting with `110...` that you will use to access the remote notebook server using an URL like:

```
https://sea.gterm.net:11001
```

Note the `https`. You will need allow your browser to use the self-signed certificate for the server.

Then type the passcode for your account that your previously saved.

Running Python

`python` Run standard python

`ipython` Run *ipython* (enhanced interative python)

`ipython notebook` Run *ipython notebook*

IPython

IPython is enhanced *interactive* shell for python http://ipython.org

Run it using the `ipython` command in the terminal

Within `ipython`, you can use the *?* command for help:

```
import math
```

```
?math.sin
```

Traditional python programming

Small programs (2-4 lines) can be typed into the terminal

Larger programs are saved in *text* files, with extension `.py`

Editor/terminal workflow

- Using a text editor, express each step as a Python statement to create a program
- Use `python textfile.py` to run the program to display the text output
- If this generates error messages or the wrong output, modify the program, save it, and run it again.
- Repeat until the correct text output is displayed by the program (*debugging*)

Editor/terminal/graphics workflow

This is similar to the Editor/terminal workflow, but allows graphical output to be displayed in a separate window.

More cumbersome to use on a remote computer. Need to use X forwarding to display graphics locally

---

Running programs using ipython notebook

IPython Notebook is a web-based interface for IPython that displays code and output in a single browser window

To use it, `cd` to your Desktop folder (or any other directory where you wish to store notebooks) in the terminal

Start the notebook server using the following command:

```
ipython notebook
```

This should also open up a Notebook window using your default browser.

You can also connect to the server in any browser using the URL http://localhost:8888

Type *Control-C* to stop the server

Using IPython Notebook on Windows

If you have installed the Anaconda distribution, the IPython Notebook server can be launched from the `Anaconda` folder in the `Start` menu. This will open up a command prompt window and run the server. Kill the command prompt window to terminate the server.

To easily open downloaded files in the notebook, save downloaded files to the `Documents\IPython Notebooks` folder.

Notebook workflow

Uses a single window!

- Type program in Code cell
- Type Control-Enter to run it
- If correct output is not displayed below the Code cell, edit program and run it again.
- Repeat until the correct output is displayed
- Type Shift-Enter to run code in current cell and then create a new cell
- Annotations can be added by inserting a Markdown cell where needed
- You can type the TAB key after the opening parantheses of a function name to obtain help

---

Notebook cell types

1. Code
2. Markdown (annotation)
3. Raw (passthrough)

Shell magic for Unix commands in code cells

%cd

%pwd

%ls

%automagic

Automagic allows shell commands to be used without the `%` prefix

---

Debugging within notebook cells

```
%debug
a = 2
b = 0
c = a/b
```

Graphical output in the notebook

`matplotlib` is powerful 2D plotting library for python

`pylab` is a collection of command style functions that make matplotlib work like MATLAB

Use `%pylab` *magic* command to import it

```
%pylab inline
pylab.plot([1, 5, 9, 16, 25])
pylab.ylabel('some numbers')
```

Edit mode vs. command mode

Use Edit more to enter and edit code.

Use Command mode to navigate and manipulate cells

Esc to enter command mode (gray boxes) *h* to list keyboard shortcuts *Enter* to enter edit mode (green boxes)

Tab completion in edit mode

TAB: variable/function name completion

Shift TAB: Function help information

A brief introduction to Markdown

Markdown is an extremely *readable* markup language. HTML (or XML) are more powerful, but are not readable.

Markdown example:

```
## This is a heading line

You can *italicize* or **bold** words.

Create lists easily

* Apple

* Bananas

* Oranges

You can include code snippets like this ``python -i option``
or by indenting 4 spaces

    # Example code
    print "this string"
```

Advanced Markdown

Markdown allows inline HTML

```
This is a regular paragraph.

<table>
    <tr>
        <td>Foo</td>
    </tr>
</table>

This is another regular paragraph.
```

MathJAX: TeX within the notebook

```
**Introduction**

*Lorenz Equations* are

\begin{align}
\dot{x} & = \sigma(y-x) \\
\dot{y} & = \rho x - y - xz \\
\dot{z} & = -\beta z + xy
\end{align}

*Maxwell's Equations* are

\begin{align}
\nabla \times \vec{\mathbf{B}} -\, \frac1c\, \frac{\partial\vec{\mathbf{E}}}{\partial t} & = \frac{4\pi}{c}\vec{\mathbf{j}} \\   \nabla \cdot \vec{
\nabla \times \vec{\mathbf{E}}\, +\, \frac1c\, \frac{\partial\vec{\mathbf{B}}}{\partial t} & = \vec{\mathbf{0}} \\
\nabla \cdot \vec{\mathbf{B}} & = 0
\end{align}
```

NumPy: Numerical array processing

`numpy` is python module for processing multidimensional arrays.

A list is a 1-dimensional array.

A matrix is a 2-dimensional array (e.g., an Excel spreadsheet)

Example of a 2-dimensional array:

```
[ [1., 2., 3.],
  [4., 5., 6.] ]
```

The *shape* of an array refers to the dimensions of the array. E.g., the above array has the shape `(2, 3)`, i.e., 2 rows and 3 columns.

Creating an array

```
import numpy as np
arr = np.array([ [1, 2], [3, 4] ])

arr.ndim               # No. of dimensions (attribute)
arr.shape              # Shape of the array
arr.size               # Size of the array
arr.dtype              # Array data type
```

Graphical output within the notebook

`pylab` is a Python module that allows MATLAB-plotting.

```
%pylab inline

pylab.plot([1, 5, 9, 16, 25])
pylab.ylabel('some numbers')
pylab.show()
```

Creating a simple plot of a cubic function

```
x = np.linspace(-10, 10, 21)

print x

y2 = x**2

print y

y3 = x**3

pylab.plot(x, y3)

pylab.plot(x, y2, "--r")
```

Plotting a sine-wave

Using `ipython notebook`

(a) Create a `numpy` array `x` containing 91 equally spaced angle values between 0 and 360 degrees using the `np.linspace` function.

(b) Create another `numpy` array `y` containing `sin` values for each `x` value. Remember to convert the angle from degrees to radians before the sine value (radians = degrees * 3.1416/180.0)

(c) Plot `y` vs. `x`, including a plot title and axis labels, using a red dashed line style.

(d) Check that the code works. Then copy and paste it as your answer. (Use `print` statements to debug your code.)

Plotting sine wave using NumPy (answer)

```
x = np.linspace(0., 360., 91, endpoint=True)  # Generate uniformly spaced 1-d array
y = np.sin(x*np.pi/180.)                      # Compute sin(x)
pylab.plot(x, y, "--r", linewidth=2.5)        # Dashed red line

pylab.xlim(0, 360.)      # Set X-axis limits
pylab.ylim(-1.5, 1.5)    # Set Y-axis limits
```

Write python code in a cell to plot both `sin(x)` and `cos(x)` for `x` ranging from 0 to 360 degrees in the same plot, using lines of different style/color.

NetCDF file format

Network Common Data Format

Metadata: information about data such as names of variables, units etc.

Example: Titles and column headers in a text file

- No standard format (or protocol) for exhanging header information

A NetCDF file combines metadata and data in a binary format

Installing netCDF4 in Anaconda

If using a Mac/Linux system, open a terminal. If using Windows, run the program `cmd` to open a console window. (You can do this by searching for the program named `cmd`)

Type the command `conda install netCDF4` in the terminal/console window.

Simple plotting in the notebook from netCDF file

The input data file is `standard_atmosphere.txt` which contains standard atmospheric properties as a function of altitude. You can download it from the link http://pyintro.org/static/atmo321/data/standard_atmosphere.txt

Download it and save it to the folder where you are running the notebook server.

```
wget http://pyintro.org/static/atmo321/data/standard_atmosphere.nc
```

```
OR
```

```
wget http://pyintro.org/static/atmo321/data/standard_atmosphere.nc
```

```
ncdump -h standard_atmosphere.nc
```

On Windows, download the file using the browser from http://pyintro.org/blog/pages/data.html and save the file to the *Documents\IPython Notebooks* folder.

Opening a netCDF file

```
import netCDF4
```

```
ncfile = netCDF4.Dataset('standard_atmosphere.nc')  # Open file
```

```
print ncfile.dimensions        # All Dimension objects in file
```

```
print ncfile.variables         # All Variable objects in file
```

```
print ncfile.variables.keys()  # All Variable names in file
```

Extracting a single variable from a netCDF file

```
T = ncfile.variables["temperature"]   # Get Variable object for "temperature"
```

```
print type(T)
```

```
T_data = T[:]          # Extract NumPy data array from Variable
```

```
print type(T_data)
```

```
print T.ncattrs()      # List all variable attributes
```

```
T.units                # Units attribute
```

Write some python code that creates an inline plot of the temperature as a function of altitude using data from the file `standard_atmosphere.nc`.

Read netCDF and plot (answer)

```
import netCDF4

ncfile = netCDF4.Dataset('standard_atmosphere.nc')  # Open file

print ncfile.variables.keys()  # All Variable names in file

T = ncfile.variables["temperature"]       # Get Variable object for "temperature"
altitude = ncfile.variables["altitude"]   # Get Variable object for "altitude"


T_data = T[:]          # Extract NumPy data array from Variable
```

```
altitude_data = altitude[:]

pylab.plot(T_data, altitude_data)

pylab.xlabel(T.units)
pylab.ylabel(altitude.units)
pylab.title("T vs. altitude")
```

Components of a netCDF file

Dimensions: Data dimensions, suc as `longitude, latitude, level, time, ...`

Variables: Data with attributes such as `units`, `long_name`, ...

- *Coordinate variables*: one-dimensional variables with the same name as a dimension
- *Data variables*: Other variables with one or more dimensions

Attributes: May be attached to variables, or be global

A more complex netCDF file

```
wget http://pyintro.org/static/atmo321/data/air.mon.ltm.nc

    OR

curl http://pyintro.org/static/atmo321/data/air.mon.ltm.nc >  air.mon.ltm.nc # Download data file

ncdump -h air.mon.ltm.nc      # Display header (metadata) information for file

ncdump -c air.mon.ltm.nc      # Display header and coordinate information for file
```

Opening a netCDF file

```
ncfile = netCDF4.Dataset('air.mon.ltm.nc')  # Open file

print ncfile.dimensions       # All Dimension objects in file

print ncfile.variables        # All Variable objects in file

print ncfile.ncattrs()        # All global Attributes

# Properties of variables
for varname in ncfile.variables:
    print varname, ncfile.variables[varname]
```

Extracting a single variable from a netCDF file

```
air_temp = ncfile.variables["air"]   # Get Variable object for "air"

print type(air_temp)

temp_data = air_temp[:]       # Extract NumPy data array from Variable

print type(temp_data)

print air_temp.ncattrs()     # List all variable attributes

air_temp.units               # Units attribute
```

Contour plots with matplotlib

- `contour(x, y, z, n)` for contour lines
- `contourf(x, y, z, n, cmap=...)` for filled contours
- `clabel(...)` for contour labels
- `colorbar(...)` for color bar

Creating a contour plot (modified from Johnny Lin's textbook)

```
%pylab inline
import netCDF4
ncf = netCDF4.Dataset("air.mon.ltm.nc", "r")
lon = ncf.variables["lon"][:]
```

```
lat = ncf.variables["lat"][:]
temp_data = ncf.variables["air"][0,0,:,:]
temp_data.shape

pylab.contourf(lon, lat, temp_data, 10, cmap=cm.rainbow)
pylab.axis([0, 360, -90, 90])
pylab.colorbar(orientation="horizontal")

cont = pylab.contour(lon, lat, temp_data, 10)   # Save object
pylab.clabel(cont)
```

Vertical section plot

```
%pylab inline
import netCDF4

ncf = netCDF4.Dataset("air.mon.ltm.nc", "r")
level_data = ncf.variables["level"][:]
lat_data = ncf.variables["lat"][:]

# Extract data for month_index 0, all levels, all lats, lon_index 0
temp_data = ncf.variables["air"][0,:,:,0]
temp_data.shape

pylab.contourf(lat_data, level_data, temp_data, 10, cmap=cm.rainbow)
pylab.ylim(1000, 0)
pylab.colorbar()

pylab.contour(lat_data, level_data, temp_data, 10)
```

Interactive notebook widgets

- Intro: http://nbviewer.ipython.org/github/esss/ipython/blob/master/examples/Interactive%20Widgets/Index.ipynb
- Contour plotting examples: http://earthpy.org/pyncview_pm.html

---

Interactive widget methods

`interact`: Interactive call to function

```
interact(func_name, arg1=..., arg2=...)
```

`fixed`: Fixed argument value

Widget example

```
from IPython.html.widgets import interact, fixed
from IPython.html import widgets

def add(x, y):
    return x + y

def show_add(x, y):
    print add(x, y)

interact(show_add, x=3, y=(0,9))
```

Task 1: Formatting time

In a notebook cell, write a Python function `time_stamp(hour, minute, pm=False)` that takes two numeric arguments and one boolean argument, returning a time stamp string formatted as follows:

```
print time_stamp(8, 5, pm=True)
```

should return

```
08:05 PM
```

Task 2: Interactive time entry

In another notebook cell, define a new function `show_time(hour, minute, pm=False)` as follows:

```
def show_time(hour, minute, pm=False):
  print time_stamp(hour, minute, pm=pm)
```

Use the `interact` widget method to interactively call `show_time` using a slider for hour and minutes and a checkbox for AM/PM, using appropriate ranges for hour and minute values.

Plotting sine wave interactively

```
x = np.linspace(0., 360., 91, endpoint=True)  # Generate uniformly spaced 1-d array
y = np.sin(x*np.pi/180.)                      # Compute sin(x)
pylab.plot(x, y, "--r", linewidth=2.5)

pylab.xlim(0, 360.)      # Set X-axis limits
pylab.ylim(-1.5, 1.5)    # Set Y-axis limits
```

Interactive plotting

The El Nino index (NINO3.4) is defined as the average of sea surface temperature (in degrees Celsius) over a region in the East Central Tropical Pacific (5N-5S, 170-120W). The netCDF file `elnino34.ncd` at http://pyintro.org/blog/pages/data.html contains *monthly* El Nino index values for years 1950 to 2009.

You can download it from the link http://pyintro.org/static/atmo321/data/elnino34.ncd

Save it to the folder where you will run `ipython notebook`. (On Windows, save it to he *Documents\IPython Notebooks* folder.)

Note You may need to right-click on the above file link to download it, and be sure to retain the extension â€œncdâ€.

Use the `interact` widget method to plot El Nino index values for different months from the file `elnino34.ncd`.

First, write a function `elnino_plot(month)` which takes a `month` value from 1 to 12 as an argument and plots the El Nino time series. Include the month number in the plot title.

Next, call the function using the `interact` method, so that the user can plot the El Nino index for different months simply using a slider. Use an appropriate range for month values.

Map projections

Properties: conformal, equal area, equidistant, ...

Examples

- Cylindrical (Mercator)
- Conic
- Azimuthal (Orthographic, stereographic)

Using projections and basemap

See http://matplotlib.org/basemap/users/examples.html

- `import mpl_toolkits.basemap`
- `Basemap(projection=...)` to set up the mapping
- `drawcoastlines()`
- `fillcontinents(...)`
- `drawparallels(...)` draw latitude lines
- `drawmeridians(...)` draw longitude lines
- `mapproj()` transform coordinate arrays

To install `basemap` in Anaconda, use the command

```
conda install basemap
```

Basemap tutorial

http://introtopython.org/visualization_earthquakes.html

(http://introtopython.org also has a nice tutorial for python in general)

Basemap example (from Johnny Lin's textbook)

```
import mpl_toolkits.basemap as bm
mapproj = bm.Basemap(projection='cyl', llcrnrlat=-60.0, llcrnrlon=-0.0, urcrnrlat=60.0, urcrnrlon=360.0)
mapproj.drawcoastlines()
mapproj.drawparallels(np.array([-60, -40, -20, 0, 20, 40, 60]), labels=[1,0,0,0])
mapproj.drawmeridians(np.array([-180, -90, 0, 90, 180]), labels=[0,0,0,1])
lonall, latall = np.meshgrid(lon, lat)
lonproj, latproj = mapproj(lonall, latall)
pylab.contour(lonproj, latproj, temp_data)
```

Creating overlaid color and contour plots in the notebook

Using the files `air.mon.ltm.nc` and `hgt.mon.ltm.nc` from http://pyintro.org/blog/pages/data.html write a function `tempheight(month)` that displays the 1000hPa temperature (in color) and the 500hPa geopotential height (as overlaid contours) in the domain 130W-60W, 25N-55N using cylindrical projection for the selected `month` value, ranging from `1..12`. Draw coastlines, states and countries on the map, and include axis ticks.

Run the code in `ipython notebook`. The final plot should look similar to:

https://dl.dropboxusercontent.com/u/72208800/atmo321/exercise7.png

---

Creating overlaid color and contour plots (solution)

```
import netCDF4
import mpl_toolkits.basemap as bm
import numpy as np
import matplotlib.pyplot as plt
%pylab inline

def tempheight(month):

    ncf = netCDF4.Dataset("air.mon.ltm.nc", "r")
    ncf2 = netCDF4.Dataset("hgt.mon.ltm.nc", "r")
    lat_data = ncf.variables["lat"][14:27]
    lon_data = ncf.variables["lon"][92:121]

    temp_data = ncf.variables["air"][month-1,0,14:27,92:121]
    hgt_data = ncf2.variables["hgt"][month-1,5,14:27,92:121]
    ncf.close()
    ncf2.close()

    mapproj = bm.Basemap(projection= 'cyl',llcrnrlat=25.0, llcrnrlon=230,urcrnrlat=55.0, urcrnrlon=300)
    mapproj.drawcoastlines()
    mapproj.drawcountries()
    mapproj.drawstates()
    mapproj.drawparallels(np.array([30, 40, 50]), labels=[1,0,0,0])
    mapproj.drawmeridians(np.array([-120, -100, -80]), labels=[0,0,0,1])
    mymapf = pylab.contourf(lon_data, lat_data, temp_data, 10,cmap=pylab.cm.jet)
    mymap = pylab.contour(lon_data, lat_data, hgt_data, 10,colors='k')
    pylab.colorbar(mymapf, orientation='horizontal')
    pylab.title("Month "+str(month))
    return pylab.show()

tempheight(1)
```

Creating a regional plot with basemap

```
import mpl_toolkits.basemap as bm

import netCDF4
ncf = netCDF4.Dataset("air.mon.ltm.nc", "r")
lon = ncf.variables["lon"][:]
lat = ncf.variables["lat"][:]
temp_data = ncf.variables["air"][0,0,:,:]

mapproj = bm.Basemap(projection='lcc', lat_0=30, lon_0=250, llcrnrlat=20.0, llcrnrlon=230, urcrnrlat=55.0, urcrnrlon=310)
mapproj.drawcoastlines()
lonall, latall = np.meshgrid(lon, lat)
lonproj, latproj = mapproj(lonall, latall)
pylab.contourf(lonproj, latproj, temp_data, 20)
pylab.colorbar()
```

Using `pandas` in the notebook

`pandas` is a powerful data analysis package for spreadsheet-like operations in Python.

We will work through a simple example from *Wes McKinney's* book *Python for Data Analysis* using IPython Notebook, which provides a nicer-lokking display of `pandas` output.

First, install `pandas`:

```
conda install pandas
```

From the Social Security Administration website http://www.ssa.gov/oact/babynames/limits.html download the National Data for popular baby names in the U.S. in the file `naames.zip`

Create sub-folder named `names` in the folder where your notebook is running. Unzip the downloaded file in the `names` sub-folder (this create a large number of files)

Reading the baby names data

```
import pandas as pd

names1880 = pd.read_csv('names/yob1880.txt', names=['names', 'sex', 'births'])

# Display the top 10 names
names1880[:10]
```

What are the 10 most popular baby names in 2013?

Combining data for all years

```
years = range(1880, 2014)
pieces = []
columns = ['names', 'sex', 'births']

for year in years:
    path = 'names/yob%d.txt' %year
    frame = pd.read_csv(path, names=columns)
    frame['year'] = year
    pieces.append(frame)
names = pd.concat(pieces, ignore_index=True)
```

Analyzing data

```
total_births = names.pivot_table('births', index='year',
                                 columns='sex', aggfunc=sum)
total_births.tail()
#total_births.plot(title='Total births by sex and year')
```