Introduction to High Performance Scientific Computing

Autumn, 2016

Lecture 6

Imperial College
London

Python notes

Getting comfortable with python:

- **Have command of *all* of the material in lecture 3 slides and numpy section of lecture 5 slides (use lab 2 and lab 3 exercises for self-assessment)**
- **Understand structure and purpose of functions (lecture 4 slides)**
- **Understand *mysqrt.py* (provided in lecture5 directory of course repo)**
- **Today's material: plotting, SVD, solving initial value problems**
- **Further help: list of supplementary material on course webpage, office hours**

Importing modules

- **In scripts, use:**

```
In [1]: import numpy as np

In [2]: np.linspace(0.0,1,3)
Out[2]: array([ 0. ,  0.5,  1. ])
```

- **This way, users know which packages are needed and which specific functions are being used**

- **If only using one or two functions, can use:**

```
In [3]: from numpy import linspace

In [4]: linspace(0.0,1,3)
Out[4]: array([ 0. ,  0.5,  1. ])
```

Importing modules

- **Can also import all functions at once:**

```
In [6]: from numpy import *

In [7]: linspace(0.0,1,3)
Out[7]: array([ 0. ,  0.5,  1. ])
```

- **Ok at terminal, but not in scripts! Makes it difficult to see where and how the module is being used.**

Importing modules

- **Can also import all functions at once:**

```
In [6]: from numpy import *

In [7]: linspace(0.0,1,3)
Out[7]: array([ 0. ,  0.5,  1. ])
```

- **Ok at terminal, but not in scripts! Makes it difficult to see where and how the module is being used.**

- **When launching ipython with** ipython --pylab**, the pylab flag means terminal is launched with:**

```
In [1]: from numpy import *
```

-

```
In [2]: from matplotlib import *
```

Importing modules

- **Can also import all functions at once:**

```
In [6]: from numpy import *

In [7]: linspace(0.0,1,3)
Out[7]: array([ 0. ,  0.5,  1. ])
```

- **Ok at terminal, but not in scripts! Makes it difficult to see where and how the module is being used.**

- **When launching ipython with** ipython --pylab**, the pylab flag means terminal is launched with:**

```
In [1]: from numpy import *

In [2]: from matplotlib import *
```

- **The** %pylab **command in ipython notebook does the same thing.**

2d plots

- **Matplotlib package provides Matlab-like plotting**
- **Usually included in scripts as:** `import matplotlib.pyplot as plt`
- **Will look at illustrative example here and provide supplementary ipython notebook**

2d plots: simple example

- **Create and plot 2 simple functions**

```
import numpy as np
import matplotlib.pyplot as plt

#Create some arrays to be plotted
Nx = 100
Ny = 200
x = np.linspace(0.0,np.pi,Nx)
y = np.linspace(-np.pi,np.pi,Ny)

f = np.sin(x)
g = np.cos(y)
```

2d plots: simple example

- **Create and plot 2 simple functions**

```
import numpy as np
import matplotlib.pyplot as plt

#Create some arrays to be plotted
Nx = 100
Ny = 200
x = np.linspace(0.0,np.pi,Nx)
y = np.linspace(-np.pi,np.pi,Ny)

f = np.sin(x)
g = np.cos(y)

#Create plot
plt.figure() #make new figure

plt.plot(x,f,'b-',label='sin') #blue line
plt.plot(y,g,'r--',label='cos') #red dashed line
```

2d plots: simple example

- **Create and plot 2 simple functions**

```
#Create plot
plt.figure() #make new figure

plt.plot(x,f,'b-',label='sin') #blue line
plt.plot(y,g,'r--',label='cos') #red dashed line

#add axis labels,legend, and figure title
plt.xlabel('time')
plt.ylabel('f(t),g(t)')
plt.legend(loc='best')
plt.title('Illustrative figure prepared by Prasun Ray')
```

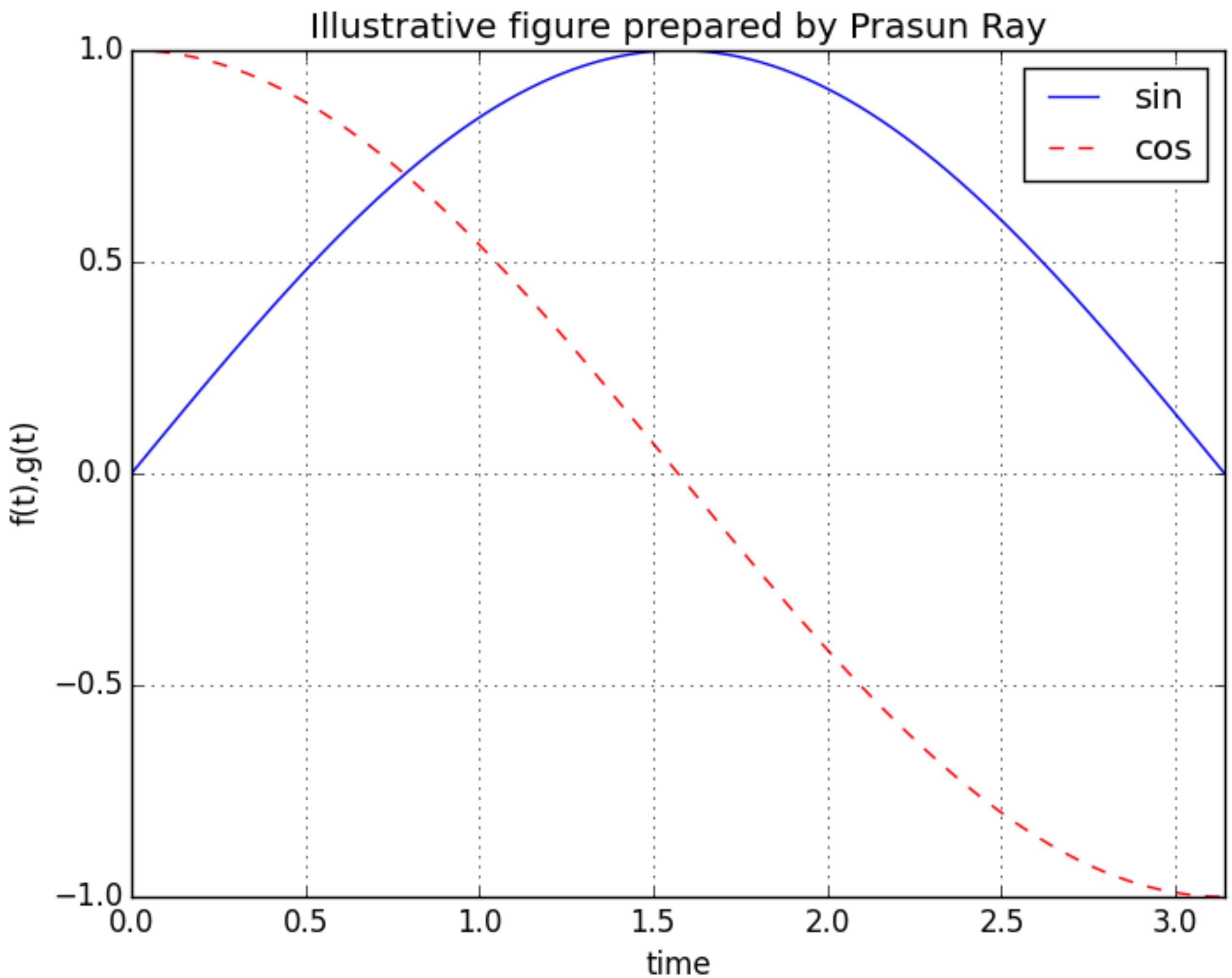
2d plots: simple example

- **Create and plot 2 simple functions**

```
#Create plot
plt.figure() #make new figure

plt.plot(x,f,'b-',label='sin') #blue line
plt.plot(y,g,'r--',label='cos') #red dashed line

#add axis labels,legend, and figure title
plt.xlabel('time')
plt.ylabel('f(t),g(t)')
plt.legend(loc='best')
plt.title('Illustrative figure prepared by Prasun Ray')

#adjust x-axis limits, turn on grid, display and save figure
plt.xlim(0,np.pi)
plt.grid()
plt.show()

plt.savefig('plot_example.png')
```

2d plots: simple example

- **Create and plot 2 simple functions**
- ***plot_example.png*:**



2d plots

- **Use** loglog, semilogx, semilogy **for logarithmic axes**
- contour **for functions of two variables**
- hold(True) **or** hold(False) **to overlay curves on single figure (or not)**
- **Example code in repo:** ***plot_example.py***
- **See online tutorial for further info: http://matplotlib.org/users/pyplot_tutorial.html**
- **Also look at: http://matplotlib.org/gallery.html (includes complex figures + code that generates them)**

scipy overview

- *scipy* is a module which contains a wide variety of scientific tools
- A few useful submodules:
  - *scipy.special*
  - *scipy.integrate*
  - *scipy.optimize*
  - *scipy.fftpack*
  - *scipy.signal*
- Try tab completion: scipy. <tab>
  import scipy. <tab>

A scipy example

- **Use** *odeint* **from** *scipy.integrate* **module to solve:**

$$\frac{d^2y}{dt^2} + \omega^2 y = 0$$

- **First, rewrite as two 1st-order ODEs:**

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -\omega^2 y_1$$

A scipy example

$$\frac{dy_1}{dt} = y_2$$

$$\frac{dy_2}{dt} = -\omega^2 y_1$$

- **Basic idea: discretize time,** t = 0, dt, …, N*dt**, and starting from** y(0) **march forward in time and compute** y(dt), … y(N*dt)

- *odeint* **chooses the stepsize,** dt, **so that error tolerances are satisfied**

- **Need to specify:**
  - **Initial condition**
  - **Timespan for integration**
  - **A Python function which provides RHS of the ODE to odeint**

- **Look at** *ode_example.py*

Singular Value Decomposition

SVD is a powerful tool for data analysis and optimization

- **Widely used for extracting "important" components of multidimensional data, reducing number of dimensions: Principal Component Analysis**
- **Provides information on maximum growth of linear ODEs**
- **Can be used for simple data compression**

Singular Value Decomposition

Overview, *any* M x N matrix, A, can be decomposed as:

$$A = USV^T$$

U **is a M x M matrix whose columns are the eigenvectors of** AA^T

V **is a N x N matrix whose columns are the eigenvectors of** $A^T A$

S **is a M x N matrix with *min*(M,N) entries on its diagonal**

- **These entries are real, non-negative, ordered**

$$S_{11} \geq S_{22} \geq S_{33} \geq ... \geq 0$$

- **Called singular values, square root of eigenvalues of** AA^T**,** $A^T A$

Singular Value Decomposition

Overview, *any* M x N matrix, A, can be decomposed as:

$$A = USV^T$$

Importance of SVD stems (primarily) from importance of AA^T **and** A^TA

Can be used to find maximum of $|Ax|^2 = x^T A^T A x$

Singular Value Decomposition

Overview, *any* M x N matrix, A, can be decomposed as:

$$A = USV^T$$

Importance of SVD stems (primarily) from importance of AA^T and A^TA

Can be used to find maximum of $|Ax|^2 = x^T A^T A x$

SVD of measurements can be used to analyze *covariance* matrix AA^T :

$$A = \begin{bmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix} \qquad \frac{1}{n}\sum_{j=1}^{n} a_j = 0$$

Principal component analysis (PCA)

Singular Value Decomposition

Can rewrite SVD as:

$$A = u_1 S_{11} v_1^T + u_2 S_{22} v_2^T + u_3 S_{33} v_3^T + \ldots$$

Here, u_i **is i^th eigenvector of** AA^T

and v_i **is i^th eigenvector of** $A^T A$

- **These eigenvectors have length=1, so the singular values indicate importance of each term in sum**
- **Can discard terms with small** S_{ii} **values**
- **Then only need to save the first "K" eigenvectors and singular values**
- **Can reconstruct partial sum from these stored quantities, choose K based on: how quickly** S_{ii} **terms decrease, and desired memory savings**

Singular Value Decomposition

SVD in Python:

- np.linalg.svd
- scipy.linalg.svd
- scipy.sparse.linalg.svds

- scipy.linalg **routines will be at least as fast as** np.linalg**, depending on installation, could be faster**
- sparse.linalg.svds **slower than others, but uses *much* less memory.**
  -- Very important for large matrices (and datasets)

Beyond Scipy and Numpy

Pandas: Powerful module for data analysis

Scikit-learn: Machine learning

And much more… look around online!

Homework 2

- Will be posted late afternoon today
- Using python for image processing
- Numerically solve system of ODEs modeling spread of infectious diseases
- Today’s office hour (in MLC): 4-5 pm instead of 5-6 pm