A quick intro to Python

James J. Shepherd

Massachusetts Institute of Technology

July 11, 2017

- I am a postdoc at MIT
- Thanks to Troy Van Voorhis & group at MIT

- (I start at the University of Iowa in mid-August)

Python will be used throughout this week.

In the next two hours, we will...

Part 1

- Introduce libraries we will be using throughout
- Quick review some basic functions

Part 2

- Exercises to solve in groups

Before I begin...

Who has used Python before?

Before I begin...

Who has used Python before?

What kinds of things can you do with it?

Before I begin...

Who has used Python before?

What kinds of things can you do with it?

Discuss in your groups, and return examples
e.g.

Before I begin...

Who has used Python before?

What kinds of things can you do with it?

Here are some of the things I’ve used it for:

- Programming
- Data analysis
- Graphical analysis
- Organizing files
- etc.

What this session *isn't*

This is not going to be an exhaustive overview of Python

- Reason 1: That would take too long
- Reason 2: I use Python fairly pragmatically for QMC research

What this session *isn't*

This is not going to be an exhaustive overview of Python

I encourage you to find find information from:

- Expertise within your groups
- Us!
- Searching the web (Google & stackoverflow)
- Documentation (docs.scipy.org)

What this session *isn't*

This is not going to be an exhaustive overview of Python

I encourage you to find find information from:

- Expertise within your groups
- Us!
- Searching the web (Google & stackoverflow)
- Documentation (docs.scipy.org)

I am going to try to focus on themes which will be repeated over the coming week.

Part 1: Libraries are an essential part of Python for scientific computing

We are going to use 5 libraries

- `numpy`
  - is a nice demonstration for what a library is
  - module: `numpy` – essentials for scientific computing (numpy.org)

We are going to use 5 libraries

- `numpy`
  - is a nice demonstration for what a library is
  - module: `numpy` – essentials for scientific computing (numpy.org)
  - class: `numpy.ndarray` – a 'multidimensional container of items of the same type and size' (e.g. vector of integers)

We are going to use 5 libraries

- `numpy`
  - is a nice demonstration for what a library is
  - module: `numpy` – essentials for scientific computing (numpy.org)
  - class: `numpy.ndarray` – a 'multidimensional container of items of the same type and size' (e.g. vector of integers)
  - function: `numpy.array()` — turns a list into an array

We are going to use 5 libraries

- `numpy`
  - is a nice demonstration for what a library is
  - module: `numpy` – essentials for scientific computing (numpy.org)
  - class: `numpy.ndarray` – a ‘multidimensional container of items of the same type and size’ (e.g. vector of integers)
  - function: `numpy.array()` — turns a list into an array
  - method: numpy.ndarray.max – finds max of array

We are going to use 5 libraries

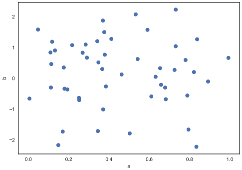
- `numpy`
  - is a nice demonstration for what a library is
  - module: `numpy` – essentials for scientific computing (numpy.org)
  - class: `numpy.ndarray` – a ‘multidimensional container of items of the same type and size’ (e.g. vector of integers)
  - function: `numpy.array()` — turns a list into an array
  - method: numpy.ndarray.max – finds max of array
  - etc.
  - powerful N-dimensional array object (`ndarray` or `array`)
  - linear algebra, Fourier transform, and random number capabilities

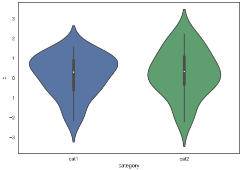
We are going to use 5 libraries

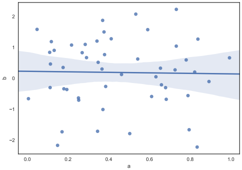
- `numpy` — fundamentals for scientific computing (numpy.org)
- `scipy` — augments `numpy` functionality (scipy.org)
  - *more* linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.
- `pandas` — data manipulation (pandas.pydata.org)

We are going to use 5 libraries

- `numpy` — fundamentals for scientific computing (numpy.org)
- `scipy` — augments `numpy` functionality (scipy.org)
- `pandas` — data manipulation (pandas.pydata.org)
- `matplotlib` — a 2D plotting library (matplotlib.org)
- `seaborn` —interfaces with `matplotlib` for drawing attractive statistical graphics (seaborn.pydata.org)







simple `numpy` example

- open python from terminal, load numpy

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

simple numpy example

```
~$ python
>>> import numpy as np
```

- What does this achieve?

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

simple numpy example

```
~$ python
```

```
>>> import numpy as np
```

- numpy commands are loaded into memory as `np`

```
>>> from numpy import *
```

- you'll also see this alternative, what's the difference?

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

simple numpy example

```
~$ python
```

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

- a is an array that derives from a list `[2,3,4]`
- `ndarray` in general can be multidimensional

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

simple numpy example

```
~$ python
```

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
```

- `ndarray.dtype` is an object for the data type of the array's elements
- what's the result here?

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

simple numpy example

```
~$ python
```

```
>>> import numpy as np
>>> a = np.array([2,3,4])
>>> a
array([2, 3, 4])
>>> a.dtype
dtype('int64')
```

- `ndarray.dtype` returns one type

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

simple numpy example

```
~$ python
```

```
>>> import numpy as np
>>> b = np.array([[1.2, 3.5, 5.1],[1.2, 3.5, 5.1]])
```

- What do you expect the output for b is?

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

simple numpy example

```
~$ python
```

```
>>> import numpy as np
>>> b = np.array([[1.2, 3.5, 5.1],[1.2, 3.5, 5.1]])
>>> print(b)
[[ 1.2  3.5  5.1]
 [ 1.2  3.5  5.1]]
>>> b.shape
(2, 3)
```

- b is a multidimensional array
- `print` and `np.shape` allow us to see this

https://docs.scipy.org/doc/numpy-dev/user/quickstart.html

Examples to note!

array creation

- `np.random.randn` – normally distributed random numbers
- `np.zeros` – an array of zeros
- `np.newaxis` – increase dimension of array

simple math

- `np.sqrt` – square root
- `np.exp` – exponential function
- `np.outer` – outer product for vectors

statistics

- `np.sum` – sum of elements
- `np.mean` – average
- `np.std` – standard deviation

simple plotting example

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

- load modules

simple plotting example

```
npts=50
df={'a':np.random.random(npts),
    'b':np.random.randn(npts),
    'category':['cat1']*int(npts/2) +
     ↪  ['cat2']*int(npts/2) } #Using list generation
df=pd.DataFrame(df)
```

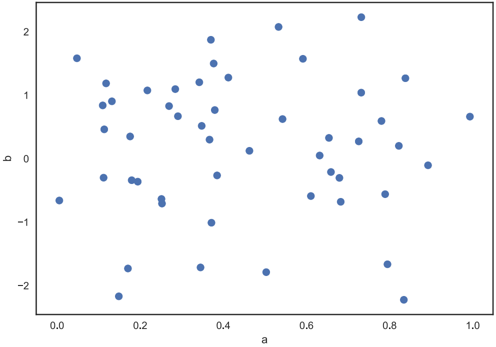
- make the data set as a pandas data frame

```
>>> print(df)
           a         b category
0   0.643472  1.073994     cat1
1   0.030219 -0.807244     cat1
...
48  0.972583 -0.153075     cat2
49  0.469095 -0.481806     cat2
```

simple plotting example

```
plt.figure()
plt.scatter('a','b',data=df)
plt.xlabel('a'); plt.ylabel('b')
plt.savefig("scatter.pdf",bbox_inches='tight')
```

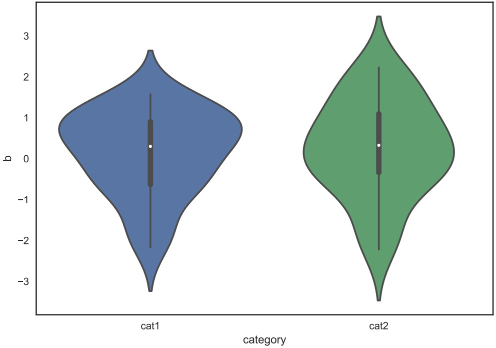
- `plt.scatter` produces a simple scatter plot



simple plotting example

```
plt.figure()
sns.set_style("white")
sns.violinplot(x='category',y='b',data=df)
plt.savefig("conditional.pdf",bbox_inches='tight')
```

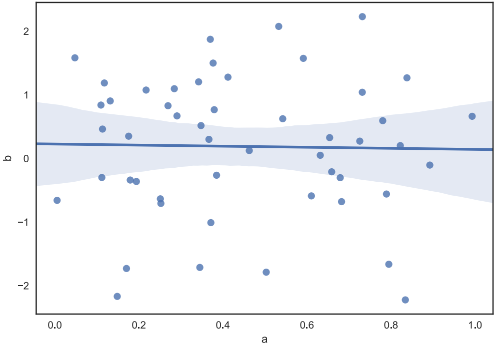
- `sns.violinplot` produces a combination of a boxplot and a kernel density estimate



simple plotting example

```
plt.figure()
sns.regplot(x='a',y='b',data=df)
plt.savefig("regression.pdf",bbox_inches='tight')
```

- `sns.regplot` plots the data with a linear regression model fit



Examples to note!

plot creation

- `plt.figure` – creates a canvas
- `plt.plot` – produces a plot of bivariate data
- `plt.scatter` – scatter plot
- `plt.subplots` – return a subplot axes in a grid

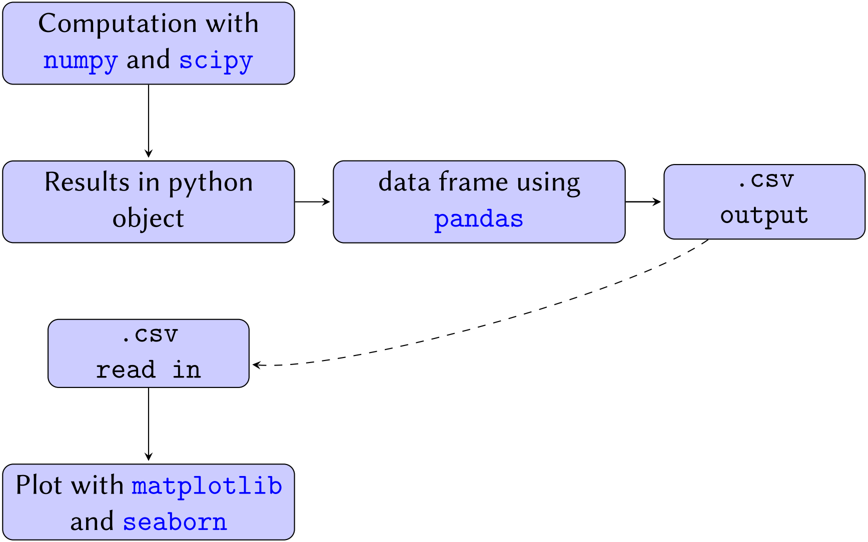
simple math

- `plt.xlabel` and `plt.ylabel` – set axis labels
- `plt.xlim` and `plt.ylim` – set axis limits

output

- `plt.savefig` – save file
- `plt.show` – see the plots on screen

Recommended workflow



Part 2: Coding in groups

- Short exercises to try to understand how to solve problems using Python
- We will also be looking at ways to test/extend ideas

The exercises

- Exercise 1: Compare exponential evaluation of a random number (interval [1,2]) with a truncated Taylor series
- Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix
- Exercise 3: Monte Carlo evaluation of π

Exercise 1: Compare exponential evaluation of a random number (interval [1,2]) with a truncated Taylor series

- Requires: `numpy` → `np`; `scipy` → `sp`; `scipy.misc` explicitly*

$$\exp(x) \approx \sum_{n}^{\mathbf{N}} \frac{1}{n!} x^n$$

Generate random `np.random.rand()`

For Taylor series, use `sp.misc.factorial(i)` and `np.power(x,i)`

Can use `np.sum(array)` and `np.multiply(array1,array2)`

Obtain comparison between $\exp(x)$ and Taylor series

- `print` your estimate; how does this change with $\mathbf{N}$?

Exercise 1: Compare exponential evaluation of a random number (interval [1,2]) with a truncated Taylor series

```
import numpy as np
import scipy as sp
import scipy.misc
x=np.random.rand()+1.0
d=np.arange(10)
c=1.0/sp.misc.factorial(d)
d=np.power(x,d)
estimate=np.sum(np.multiply(c,d))
```

```
>>> print(x,np.exp(x),estimate)
1.45468633693636 4.28313979213 4.28312634182
```

Exercise 1: Compare exponential evaluation of a random number (interval [1,2]) with a truncated Taylor series

```
import numpy as np
import scipy as sp
import scipy.misc
n=10
estimates=np.arange(n,dtype='float64')
for i in range(n):
    d=np.arange(i)
    c=1.0/sp.misc.factorial(d)
    d=np.power(x,d)
    estimates[i]=np.sum(np.multiply(c,d))
```

- what do your results look like? are they what you're expecting?
- who used something different?

Exercise 1: Compare exponential evaluation of a random number (interval [1,2]) with a truncated Taylor series

```
import numpy as np
import scipy as sp
import scipy.misc
n=10
estimates=np.arange(n,dtype='float64')
for i in range(n):
    d=np.arange(i)
    c=1.0/sp.misc.factorial(d)
    d=np.power(x,d)
    estimates[i]=np.sum(np.multiply(c,d))
```

```
>>> print(estimates)
array([ 0.        ,  1.        ,  2.45468634,
 ↪  3.51274251,  ... 4.28304596])
```

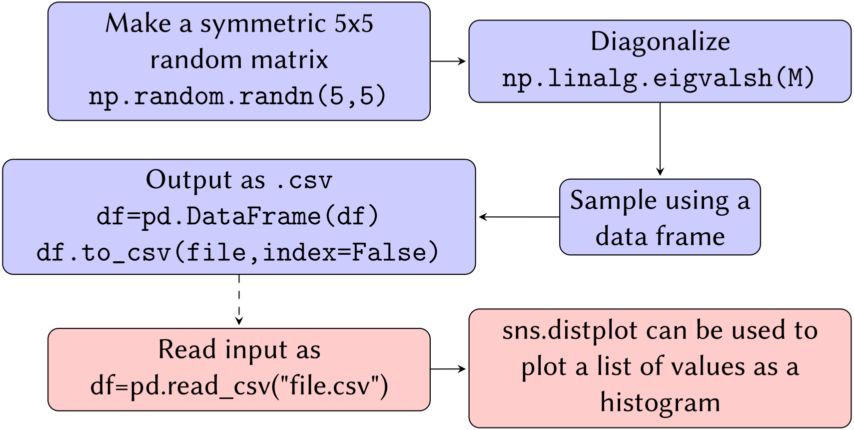
How did you test your code?

Discuss in your groups, and return examples

e.g.

Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix

- Requires: `numpy` → np; `seaborn` → sns; `pandas` → pd;



- `set_axis_labels` and `savefig` when you're done

Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix

```
import numpy as np
import pandas as pd
n=5
M=np.random.randn(n,n) # nxn matrix
M=(M+M.T)/2 # symmetrize
w=np.linalg.eigvalsh(M) #Hermetian eigenvalues
```

- Setting up the problem

```
>>> print(w)
 [-2.81657473 -1.8477462  -0.54504664
  ↪  0.39409121  1.41176684]
```

Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix

```
df={'n':[],
    'sample':[],
    'eigenvalues':[]
   } # generate empty lists
nsample=40 # the number of samples to take of each
  ↪ matrix size
for n in [5,10,20,40,100]:
  for sample in range(nsample):
    M=np.random.randn(n,n) # now an nxn matrix
    M=M+M.T # symmetrize
    w=np.linalg.eigvalsh(M) # find eigenvalues
    df['n']+=[n]*n # this makes a list n long
    df['sample']+=[sample]*n
    df['eigenvalues']+=list(w)
```

Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix

- now store df as a .csv using pandas

```
df=pd.DataFrame(df)
df.to_csv("random_matrix.csv",index=False)
```

- Can check what df looks like:

```
>>> print(df)
      eigenvalues    n  sample
0       -6.439651    5       0
1       -2.662076    5       0
2        0.647313    5       0
...
6999    27.114885  100      39
[7000 rows x 3 columns]
```

Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix

- A separate script to plot the result

```
# load modules
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix

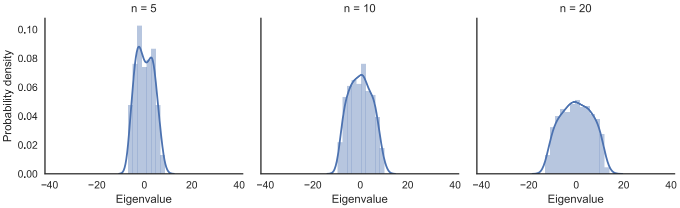
```
# load data set
df=pd.read_csv("random_matrix.csv")
```

- Can check what df looks like:

```
>>> print(df)
      eigenvalues    n  sample
0       -6.439651    5       0
1       -2.662076    5       0
2        0.647313    5       0
...
6999    27.114885  100      39
[7000 rows x 3 columns]
```

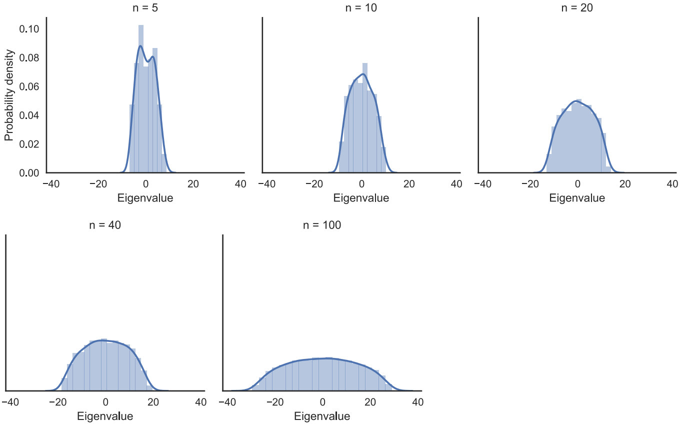
Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix

```
# make the plot
sns.set_style('white')
g=sns.FacetGrid(df,col='n')
g.map(sns.distplot,'eigenvalues')
g.set_axis_labels("Eigenvalue","Probability density")
plt.savefig("eigenvalue_distribution.pdf",
  ↪  bbox_inches='tight')
```



...

Exercise 2: Find the distribution of eigenvalues for a random symmetric matrix



- do your plots look like this?
- do these plots make sense?

Exercise 3: Monte Carlo evaluation of π

π can be written:

$$4\left(\frac{\text{area of unit circle}}{\text{area of unit square}}\right) = \pi$$

- Find π by randomly sampling points inside a square, and counting how many end up inside a circle
- (Extension) Find π by integrating a gaussian function, which is preferable?

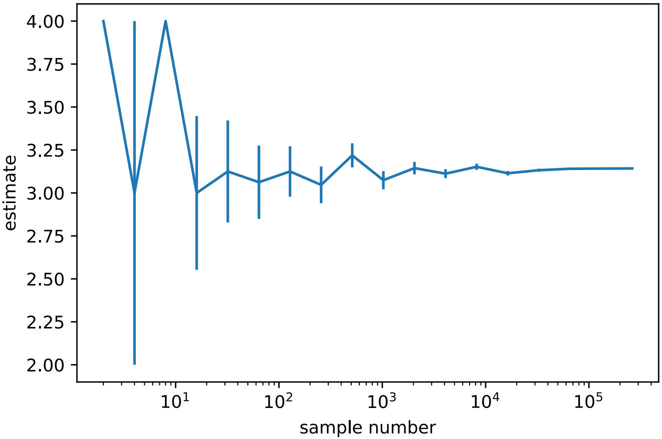
Exercise 3: Monte Carlo evaluation of π

```
import numpy as np; import scipy as sp
import scipy.linalg
import scipy.stats
import matplotlib.pyplot as plt
```

Exercise 3: Monte Carlo evaluation of π

```python
nsamples=18 # number of sample sizes
estimate=np.zeros(nsamples)*1.0 # data arrays
error=np.zeros(nsamples)*1.0
exponent=(np.arange(nsamples)+1)
for j in np.arange(nsamples)+1:
    npts=np.power(2,j) # double sampling points
    acircle=np.zeros(npts)
    pts=np.random.rand(npts,2)*2.0-1.0 # points inside unit square
    for i in range(len(pts)):
        if scipy.linalg.norm(pts[i]) < 1.0: # points inside the unit circle
            acircle[i]=1
    estimate[j-1]=np.mean(acircle)*4.0
    error[j-1]=sp.stats.sem(acircle)*4.0
```

Exercise 3: Monte Carlo evaluation of π



- not having `pandas` made this less pleasant

Concluding remarks

- Five libraries: `numpy`, `scipy`, `pandas`, `matplotlib`, and `seaborn`
- Quick data generation and problem solving