University of California, Riverside
Cosmic Dawn Center/IRES, Copenhagen University/DTU Space

Python for Astronomers

Dawn-IRES Summer 2021

Bryan Scott
bscot002@ucr.edu
Rebecca Larson and Sidney Lower

June 9, 2021

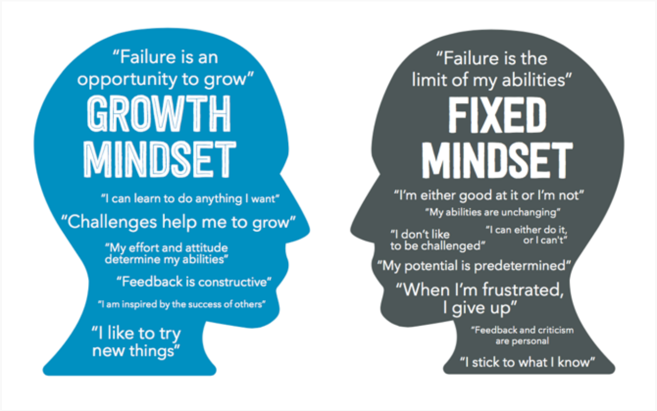
Content

Ground Rules

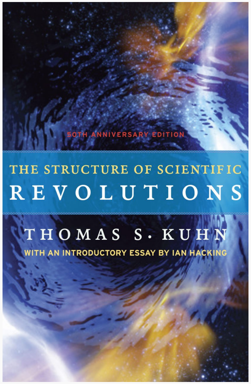
Python Workshop and IRES Program

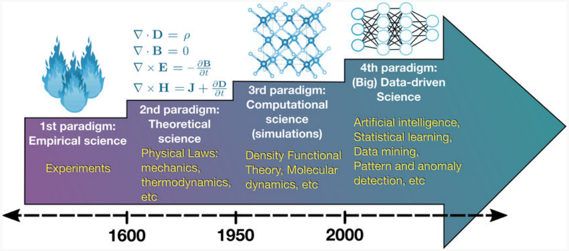
- Step up, step back
- Cultivate a safe and brave space
- Be present (virtually!)
- Honor each other’s time
- Be respectful, be responsive
- **Your ideas here**

Cultivate a Growth Oriented Mindset


"Failure is an opportunity to grow"
GROWTH MINDSET
"I can learn to do anything I want"
"Challenges help me to grow"
"My effort and attitude determine my abilities"
"Feedback is constructive"
"I am inspired by the success of others"
"I like to try new things"
"Failure is the limit of my abilities"
FIXED MINDSET
"I'm either good at it or I'm not"
"My abilities are unchanging"
"I don't like to be challenged"
"I can either do it, or I can't"
"My potential is predetermined"
"When I'm frustrated, I give up"
"Feedback and criticism are personal
"I stick to what I know"

Computing for Science? Science for Computing?


50TH ANNIVERSARY EDITION
THE STRUCTURE OF SCIENTIFIC
REVOLUTIONS
THOMAS S. KUHN
WITH AN INTRODUCTORY ESSAY BY IAN HACKING


1st paradigm: Empirical science
Experiments
$\nabla \cdot \mathbf{D} = \rho$
$\nabla \cdot \mathbf{B} = 0$
$\nabla \times \mathbf{E} = -\frac{\partial \mathbf{B}}{\partial t}$
$\nabla \times \mathbf{H} = \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t}$
2nd paradigm: Theoretical science
Physical Laws: mechanics, thermodynamics, etc
3rd paradigm: Computational science (simulations)
Density Functional Theory, Molecular dynamics, etc
4th paradigm: (Big) Data-driven Science
Artificial intelligence, Statistical learning, Data mining, Pattern and anomaly detection, etc
1600
1950
2000

Introduction
Python and Scientific Computing

- Python dominates contemporary computing in astronomy. Python has largely (but not entirely) replaced speciality tools like IDL and iraf in the work flows of most (but not all) astronomers.
- Why? Because python is a highly readable and easy to understand language. For this reason, large numbers of extensions ("packages") have been written which extend basic python functionality.
- Essential tools for Astronomy include numpy/scipy and astropy. For machine learning, scikit learn includes all of the standard ML tools, and many obscure ones. For machine learning in astronomy, the astroML package gives access to common ML and statistics tools in astronomy with large scale surveys like LSST in mind. We will focus this week on learing numpy and scipy.

What is a python package?

A python package is a folder containing

- `Files (called modules), the files will contain code that defines:`
- `Classes (objects that have properties, called attributes)`
- `Functions (or methods, that take arguments and return values)`

There are two major programming paradigms, functional and object-oriented. For the purposes of this workshop, we won't need to worry about this. That said, "prototype" code is usually written functionally, while larger projects are (almost as a rule) object oriented.

Let's write some code!

Hello World!

Before we started this workshop, you should have installed anaconda and the jupyter notebook. Open Jupyter, start a new notebook and type:

```
print('Hello World')
```

Then hit enter! (this was less a test of your coding ability, and more a test of whether software is properly installed!)

Let’s write some code!
Something More Interesting!

Let’s look at an example bit of code. Some key concepts here are the notion of a "list", floats, strings, for statements and python control statements.

```
data_list = ['-12.5', 14.4, 8.1, '42']
number_list = []
for item in data_list:
    if type(item) is str:
        item = float(item)
        number_list.append(item)
print(number_list)
```

What does this code do? How does it do it?

Let’s write some code!
Something More Interesting!

An example of a python **list** - a basic structure for holding data, including of mixed type.

```
data_list = ['-12.5', 14.4, 8.1, '42']
number_list = []
for item in data_list:
    if type(item) is str:
        item = float(item)
        number_list.append(item)
print(number_list)
```

What does this code do? How does it do it?

Let's write some code!
Something More Interesting!

The syntax for a python for loop is really simple. It just says, take every element of the list data_list and do an operation on it.

```
data_list = ['-12.5', 14.4, 8.1, '42']
number_list = []
for item in data_list:
    if type(item) is str:
        item = float(item)
        number_list.append(item)
print(number_list)
```

In this case, the operation is to take elements of the list that are strings and recast them as floats. We then store that output in a new list.

Let’s write some code!
Something More Interesting!

In this case, the operation is to take elements of the list that are strings and recast them as floats. We then store that output in a new list.

```
data_list = ['-12.5', 14.4, 8.1, '42']
number_list = []
for item in data_list:
    if type(item) is str:
        item = float(item)
        number_list.append(item)
print(number_list)
```

What is numpy?

A python package for numerical computing and linear algebra:

- The numpy **array** object is a basic datatype for most scientific workflows. "Replaces" lists in pure python.
- Similar syntax and functionality to matlab.
- Good for data oriented workflow, solving numerical problems, and general "calculations". Poor for theory. Although python symbolic manipulation packages do exist, probably best to use a computer algebra program like Mathematica.
- Why numpy? Make python faster through vectorized computation.

Vectorized Calculations

CPUs do operations that are naturally vectorized. The numpy package is written to take advantage of this.

- `Operations on lists and tuples (another data type) are slow and memory intensive.`
- `Solution: Vectorize your calculations.`
- `lists become numpy arrays (which are single data type structures`
- `use numpy versions of common mathematical operations. Numpy sqrt, trig functions, etc.`

The upshot is that by doing this, you can eliminate for loops, which are quite slow.

Array Creation

Let’s start by creating an array from a list.

```
lst = [10, 20, 30, 40]
arr = np.array([10, 20, 30, 40])
```

Now, we can "slice" the array to access its values, to get the first value of either lst or arr,

```
lst[0] = 10
arr[0] = 10
```

Note that python is a zero-indexed language. The first index is zero, which leads to lots of n-1 like things floating around.

Array Manipulation

Let's create a multi-dimensional array

```
lst2 = [[1, 2], [3, 4]]
arr2 = np.array([[1, 2], [3, 4]])
```

And let's compute the sum along the rows and columns, note the use of the axis keyword.

```
arr2.sum(axis=1) # sum along rows
arr2.sum(axis=0) # sum along columns
```

You can also do something like np.sum(arr2, axis = 0).

Array Slicing

Let's create a one-dimensional array, this creates an array with 10 values, 0-9

```
a = np.arange(10)
```

And let's extract values from these arrays,

```
a[:4] #the first 4-1 values
a[2:4] #values from index 2 to index 4-1
a[5:] #every value after index 5
a[2:9:3] #give me items 2-(9-1)
with a step size of 3
```

A Very Useful Digression into Boolean Algebra

A **boolean** is an abstract object that can take on values **True** or **False**. Define:

Logical And

A and B = AB = True iff A = True and B = True

and

Logical Or

A or B = False iff A = False and B = False

From these two statements, you can generate any other expression in Boolean logic. This proves useful for **masking** arrays.

Array Masking

Let’s create an array of random variables.

```
norm10 = np.random.normal(10, 3, 5)
```

And let’s extract the values from this array that are greater than 9.

```
# define a "mask" that we can use
# to extract values from the array

mask = norm10 > 9
print(norm10[mask])
```

The first line defines a boolean mask. It returns a list of Booleans that can be used to slice the created array. I can use logical and/or to select values that are both greater than and less than some threshold, or outside some range (greater than or less than two thresholds). Use np.where() to get the indexes where a mask would be true.

Think-Pair-Share

Pseudocode for Star Counting

One imagines that a reasonable task in astronomy is to count the number of stars in an object. Write pseudocode for doing this in two ways.

With lists and a for loop

Imagine you have a multi-dimensional list of pixel brightnesses. Write some pseudo-code to count the number of stars. (assume the PSF is unity, e.g. one pixel = one star)

You might also think about other functionality, what is the mean brightness? what is the variance in brightness?

Write pseudo-code for doing this with numpy arrays

Hint: You might be able to do this in one line. What does that line need to do? Hint2: A boolean with value True "counts" as one. A boolean with value False counts as zero.

Operations involving Multiple Arrays

Broadcasting

```
arr1 = np.arange(4)
arr2 = np.arange(10, 14)
arr_sum = arr1+arr2
```

Array multiplication by a scalar works by **broadcasting**. This means that numpy will internally reshape arrays so that they "match". The rules are a bit complicated.

```
arr_element_product = arr1*arr2
#can also np.multiply
arr_scalar_product = 5*arr1
```

Broadcasting rules: The shape of the array with fewer dimensions is padded with additional elements until it matches the other on its leading (left) side. If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other.

Operations involving Multiple Arrays

Linear Algebra

The dot product is straightforwardly

```
np.dot(arr1, arr2)
```

Matrix multiplication is

```
A = np.arange(6).reshape(2, 3)
np.dot(A.T, A)
```

np.linalg and scipy.linalg contain all of the usual linear algebra tools. np.dot can be used to quickly compute weighted sums (and related operations).

Plotting

There are several packages for making nice figures in python. The standard package/library is called **matplotlib**. A good introduction to the import statement is the following lines that you'll see at the beginning of most scientific python code:

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

The first line gives us access to all of the tools we've just covered in numpy. This line is common, but actually poor from a style POV. We should speciy what we're importing and from where it's imported from. Here, we've just grabbed everything in numpy at once. The second line is a bit better, it says we're grabbing the pyplot module from the matplotlib package. Plotting can be as easy as typing plt.plot(x,y) for x and y numpy arrays of equal length.

Matplotlib - Axes and figure

Suppose we have some data, x and y. Perhaps they were generated from x = np.linspace(0, 2*np.pi, 10), y = np.cos(x). Then we can create figure and axes objects with

```
# First, create an empty figure with 1 subplot
fig, ax1 = plt.subplots(1, 1)
```

Note, I could also use something like plt.Figure(). Using subplots allows us to generalize this to many subplots if we wanted to display a "grid" of results, a pretty common thing to do. Then to plot a couple of curves

```
# Add the lines, changing their color, style, and ma
# Black line, dashed, with 'o' markers
ax1.plot(x, y1, 'k--o', label='sin')
 # Red line, solid, with triangle-up markers
ax1.plot(x, y2, 'r-^', label='cos')
```

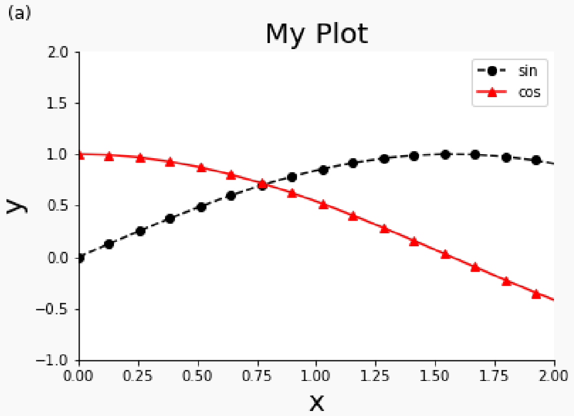
Where I've introduced a second curve, perhaps y2 = np.sin(x)

Matplotlib - Axes and figure

Then we can modify the axes to make things look nice with a few more lines of code.

```
# Adjust tick marks and get rid of 'box'
ax1.tick_params(direction='out', top=False,
right=False) # Turn ticks out
# Get rid of top axis line
ax1.spines['top'].set_visible(False)
 #  Get rid of bottom axis line
ax1.spines['right'].set_visible(False)

# Add subplot letter
ax1.annotate('(a)', (0.01, 0.96),
size=12, xycoords='figure fraction')

# Add legend
ax1.legend()
```

What does this plot look like?



Think-Pair-Share

What makes a plot a "good" plot?

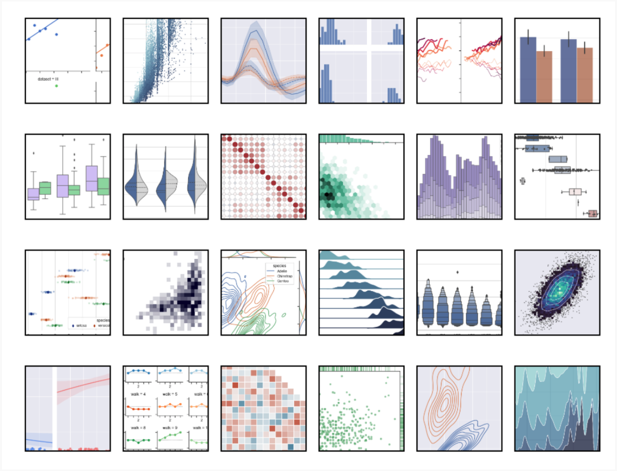
Some things to consider: Is my plot truthful? Is my plot functional? Is my plot enlightening? Is my plot insightful? Is my plot beautiful?

Of the things that make a plot "good"...

Which have to do with my plotting software?

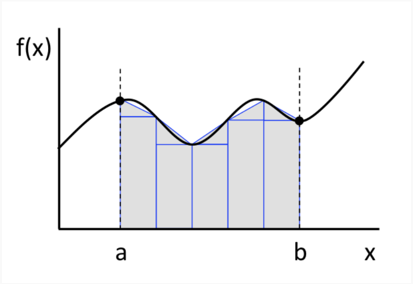
Seaborn

Seaborn is an awesome package for nice looking data-sciency plots. It’s similar to matplotlib, but makes some nicer figures with minimal additional effort over matplotlib.



Numerical Integration

The trapezoidal rule says that the area under a curve from (a,b) can be approximated by trapezoids.



It is straightforward to show that the integral is
$I = h[1/2f(a) + 1/2f(b) + \Sigma_{k=1}^{N-1} f(a+kh)]$
where N is the number of slices (and hence k labels the Nth slice), a, and b are the endpoints, and h is the width of each slice.

Defining Functions

To do the numerical integration, you will need to compute the value of the function you're integrating at a series of points. We should define a function to do this. In python, the syntax is

```
def f(x):
    return 4*x**2 + 2*x + 3
```

Note that if x is a numpy array, this function is properly vectorized.

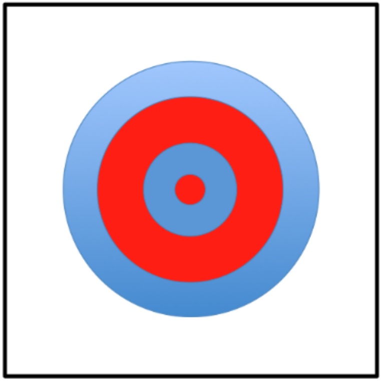
Then, the trapezoidal rule is implemented...

A properly vectorized trapezoidal rule can be done in 2-3 lines (+ variable definitions)

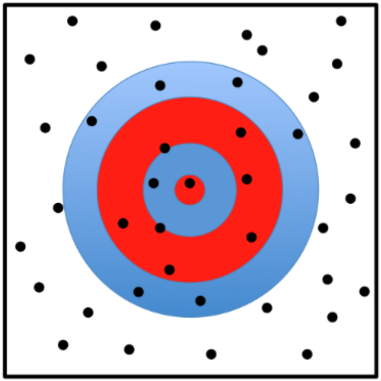
```
N = 10
a = 0.0
b = 2.0
h = (b-a)/N

k = np.arange(1,N)
I = h*(0.5*func(a) + 0.5*func(b)
+ f(a+k*h).sum())
```

Monte Carlo Methods

The classic problem - estimate the value of π through random sampling.



What is the area of the dartboard?



Throw 40 darts to find out.

Think-Pair-Share

How would you implement this in python?

What functions would you need to define? Would this be better done as a loop or a vectorized calculation?

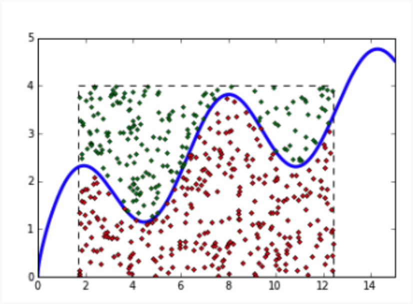
Pseudo-code for Monte Carlo Estimates of π

Pseudo Code

1. A function to randomly place points
2. A function to count how many are inside vs outside the circle
3. A function to return a MC estimate of the value of pi. This is from the fact that the ratio of area of the circle to the area of the square is $\pi/4$

Monte Carlo Integration

Estimate the Value of an Integral With the Same Technique



Monte Carlo Integration

Estimate the Value of an Integral With the Same Technique

Pseudo Code

```
1. Define a bounding box by setting some limits in x and y. This is the equivalent of the square area in the estimating $\pi$ example.
2. Randomly place points in the region.
3. Write a function to determine if a point is above or below the curve of interest.
4. Define a boolean mask that returns True if each value of $y_{sample} < f(x_{sample})$.
5. Count the true values in the mask. The ratio of the true values in the mask to the total number of points gives an MC estimate of the integral.
```

Bayesian Statistics

Bayesian statistics are an alternative to classical or "frequestist" methods. The Bayesian approach has become the dominant viewpoint in astronomy, astrophysics, and cosmology for both technical and philosophical reasons. Bayesian methods are defined by their use of Bayes' theorem

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1}$$

$P(A|B)$ - the posterior probability
$P(B|A)$ - the likelihood
$P(A)$ - the prior probability
$P(B)$ - the (Bayesian) evidence
Usually, we think of A as the parameters of some model, and B as the observed data. For example, in cosmology, we want to know how much dark matter there is (A, parameter), which we will infer from the distribution of observed CMB anisotropies (B, data).

Bayesian Statistics

Let’s suppose we have some data that we know has been generated by a Poisson process. A common example of this would be flux data. Then the likelihood is given by

$$p(k|\mu) = exp(-\mu)\frac{\mu^k}{k!} \tag{2}$$

where μ is the rate (or sometimes "location") parameter and k is the number of success/failures/photons detected. If we were working with real data, we could consider two models. One in which the rate parameter is a constant, and one where it varies linearly with redshift. Determining which model the data favors requires fully evaluated Bayes theorem, but we can estimate the parameters of the linear and constant models by simply evaluating the numerator. To do this, you should take logs of both sides, turn Bayes theorem into an addition problem (this reduced numerical errors) and sum over the numerator for each data point.

Projects for Next Week

In the github folder for this workshop, we've included two code templates. These will give you some practice with applying your new python skills to real problems you might encounter in research. Over the next week, pick and complete one of the following two projects:

1. Numerical integration of the distance-redshift relationship in an arbitrary cosmology. Complete this using a rectangle, trapezoidal, and Simpson's (quadratic) approximation to the integral. Bonus (brownie) points if you also implement a gaussian quadrature routine. Compare this with the scipy library's quad and numpy's trapz methods that you might use in your research.
2. Bayesian estimate of the Poisson likelihood. Make a 2D contour plot (using matplotlib's contour plot method) of the posterior probability distribution for linear parameters (m,b) based on a mock dataset (that you produce). Show that you recover your input parameters. Bonus points if you read up on the Maximum likelihood (classical frequentist perspective) on this problem, and discuss the differences. Why do astronomers love Bayes' theorem?

Python Libraries you should know about…

1. astropy - standard library for astronomy
2. astroml - machine learning and statistical library for astronomers, largely written for the Vera C. Rubin Observatory
3. scikit-learn - library for almost all ML applications you might need
4. seaborn - library for nice statistical visualization. More modern graphic design than matplotlib
5. pandas - R style arrays and syntax in python.
6. sympy - symbolic manipulation in python.
7. emcee - standard MCMC library for astronomers
8. Dask - an improved version of the numpy array