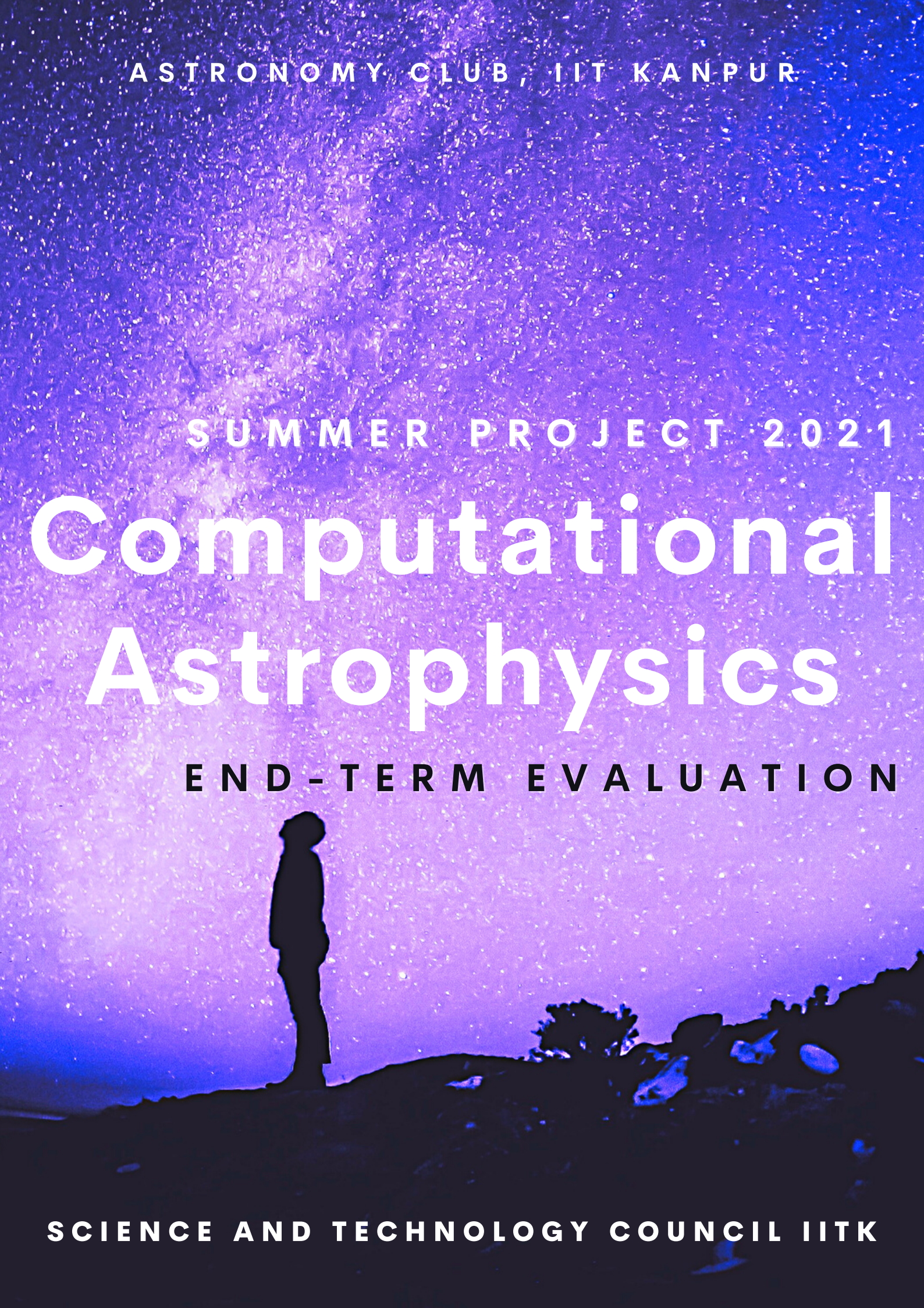
ASTRONOMY CLUB, IIT KANPUR

SUMMER PROJECT 2021

Computational Astrophysics

END-TERM EVALUATION



SCIENCE AND TECHNOLOGY COUNCIL IITK

INTRODUCTION

This project entails a simple goal: to process data and turn it into a more visual form so it can be interpreted. For this purpose, the programming language Python was adept because of its painless syntax and its built-in libraries. Starting with the basic syntax of python, we eventually built towards learning libraries, their features, sources of astronomical data, a bit of image processing and concluded by using these on the Pleiades star cluster.

CONTENTS

WEEK1: Python Basics

Abstract
The following article covers basic introduction and understanding of tools to be used in project. It covers elementary overview of Python and Jupyter Notebook, followed by getting hold onto some standard Python libraries, namely Numpy, Pandas and Matplotlib. Essence was to get well acquainted with syntax of a new language and appreciate its applicability in astrophysical domain.

[1] *Material-https://github.com/astroclubiitk/computational-astrophysics*

Contents

Week 1

1. Python Basics

1.1 Introduction

- Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Python got its name from "Monty Python's flying circus". Python was released in the year 2000.

1.2 Features of Python

- **Easy-to-learn:**- Python is clearly defined and easily readable. The structure of the program is very simple. It uses few keywords.
- **Easy-to-maintain:**-Python's source code is fairly easy-to-maintain.
- **Interpreted:**-Python is processed at runtime by the interpreter. So, there is no need to compile a program before executing it. You can simply run the program.
- **Extensible:**- Programmers can embed python within their C,C++,Java script ,ActiveX, etc.
- **Scalable:**- Python provides a better structure and support for large programs than shell scripting.
- **Object-Oriented:**- Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

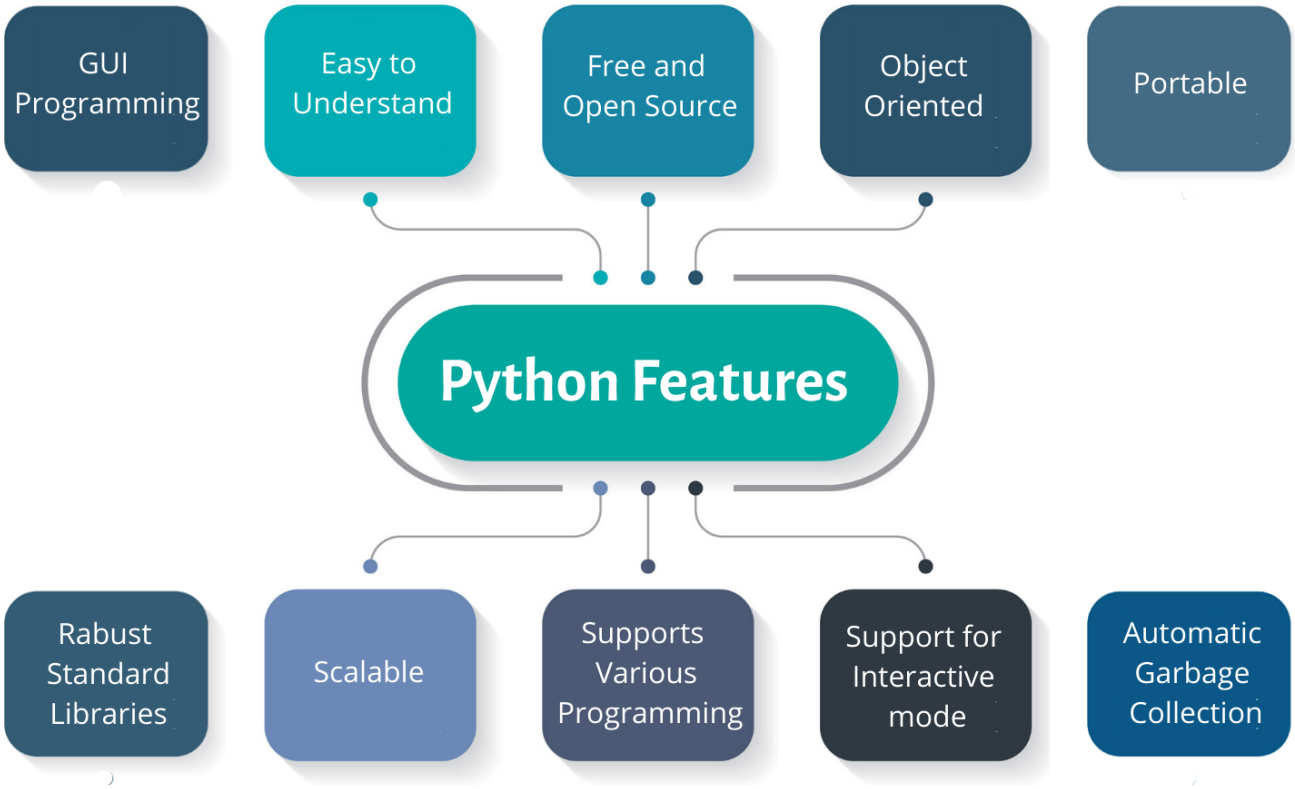


Figure 1. Python Feature

1.3 Application

- Python offers numerous options for web development. For instance, you have Django, Pyramid, Flask, and Bottle for developing web frameworks and even advanced content management systems like Plone and Django CMS.
- Python's simplicity, consistency, platform independence, great collection of resourceful libraries, and an active community make it the perfect tool for developing AI and ML applications.
- Software Developers use Python as a support language for build control, testing, and management.
- Python has a modular architecture and the ability to work on multiple operating systems. These aspects, combined with its rich text processing tools, make Python an excellent choice for developing desktop-based GUI applications.
- Python provides the skeleton for applications that deal with computation and scientific data processing. Apps like **FreeCAD** (3D modeling software) and **Abaqus**(finite element method software) are coded in Python.
- Python is also heavily used in Game Development , Enterprise and Business applications etc.

1.4 Modes of Python Interpreter

Python Interpreter is a program that reads and executes Python code. It uses 2 modes of Execution.

- **Interactive Mode:** - Interactive Mode allows us to interact with OS. When we type Python statement, interpreter displays the result(s) immediately.
- **In script Mode:** - In script mode, we type python program in a file and then use interpreter to execute the content of the file. Scripts can be saved to disk for future use.

1.5 Variables in Python

A variable allows us to store a value by assigning it to a name, which can be used later.
Python Code:
INPUT:
a=10
print(a)
OUTPUT:
10

1.6 Input and Output

Input is data entered by user (end user) in the program. In python, input () function is available for input. Output is displayed to the user using print statement.

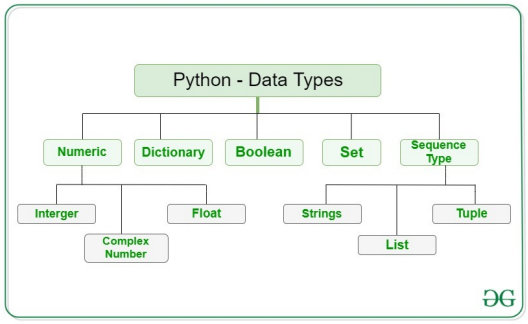
Python Code:
INPUT:
a = input(”Enter number: ”)
print(”You have entered ”, a)
OUTPUT:
Enter Number: 10
You have entered 10

1.7 Comment

Comments are parts of python code which the programmer wants to be ignored by the interpreter. A ’#’ symbol is used in the beginning of a comment.
Python Code:
INPUT:
a = ”HELLO WORLD” #This is a comment .
print(a) #This is a print statement.
OUTPUT:
HELLO WORLD

1.8 Datatypes in Python

Data types are the classification or categorization of data items. It represents the kind of value that tells what operations can be performed on a particular data. Python has 5 standard datatypes.



1.8.1 Numbers:

Number data type stores Numerical Values.This data type is immutable [i.e. values/items cannot be changed]. Python supports **integers, floating point numbers and complex numbers.**
Integers:
They are often called Integers or **int**. These are negative or positive whole numbers with no decimal point. **e.g.** 69, 3, 17 , -12 etc.
Float:
These are called **float** . These are numbers with decimal integers . **e.g.** 56.4, 23.66 etc.
Complex Numbers:
These numbers are of the form ’a + bj’. ’a’ is the Real part and ’b’ is the imaginary part and ’j’ represents square root of -1.

1.8.2 Sequence:

A sequence is an ordered collection of items, indexed by positive integers.There are three types of sequence data type available in Python.

- **Strings :** A String in Python consists of a series or sequence of characters - letters, numbers, and special characters. Individual character in a string is accessed using a subscript (index), which starts from 0. Strings are marked by quotes (” ” , ’ ’). **e.g.** ”Hello World” , ”Cat” etc.

- **Lists :** List is an ordered sequence of items. It can be written as a list of comma-separated items (values) between **square brackets [ ]**. Items in the lists can be of different data types such as **int , float , strings etc.**
  e.g. List1 = [1, 12.3, ”Giraffe”]

- **Tuple :** A tuple is same as list, except that the set of elements is enclosed in parentheses instead of square brackets. A tuple is an immutable list. Tuples can be used as keys in Dictionaries but lists cannot.
  e.g. Tup1 = (1, 12.3, ”Giraffe”)

1.9 Boolean:

Boolean data type have two values. They are 0 and 1. 0 represents False and 1 represents True.
e.g.
3==5
False

1.10 If Else Statement

Syntax:

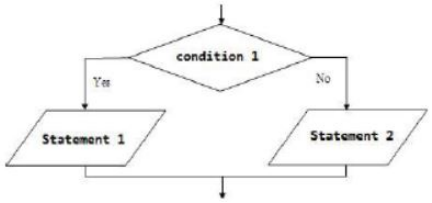


Figure 2. Flowchart of IF ELSE statement

```
    if(condition1):
    Statement
else(condition2):
    Statement2
```

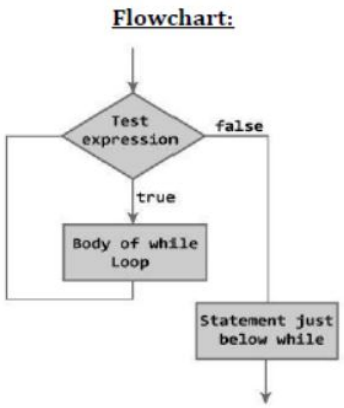
e.g.-
Python Code:
INPUT: a = float(input(”Enter Number: ”))
b = float(input(”Enter Number: ”))
if (a > b):
    print(”Greater”)
else:
    print(”Smaller”)

OUTPUT:
Enter Number: 10
Enter Number: 13.6
Smaller

1.11 While Loop

While loop in Python is used to repeatedly executes set of statement as long as a given condition is true.

Syntax:



r0.5 Initial Value
while(condition1):
    body of while loop
    increment
Exit
e.g.
Python Code:
Program to find sum of first n natural numbers:
INPUT:
n=int(input("Enter n:"))
i=1
sum=0
while(i<=n):
    sum=sum+i
    i=i+1
print("Sum of first", n , "natural number is:", sum)
OUTPUT:
Enter n: 10
Sum of first 10 natural numbers is : 55

2. Numpy

2.1 Introduction

NumPy package is imported (usually under the np alias) using the following syntax:] Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. It also contains functions for mathematical computations such as linear algebra, Fourier transform, matrices,random number capabilities etc. Along with being speed and memory efficient, NumPy works well with major libraries such as SciPy, MatPlotLib, Pandas.

NumPy package is imported (usually under the np alias) using the following syntax:

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

Figure 3. Importing Numpy

2.2 Numpy Arrays

2.2.1 Creation

Example: Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6.]The array object in NumPy is called ndarray. In NumPy, dimensions are called axes and the number of axes is called rank. We can create a NumPy ndarray object by using the array() function.

Example: Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6.

```
A=np.array([[1,2,3],[4,5,6]])
print(A)
```

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

Figure 4. Creating Array

- NumPy also offers several functions to create arrays with initial placeholder content. For example: np.zeros, np.ones, np.full, np.empty, etc.

```
distance=np.zeros((96,1),dtype=np.float32).reshape(96,1)
k=0
while(k<96):
    distance[k,0] = 10**logd[k,0]
    k=k+1
print(distance)
```

Figure 5. Creating np.zeros initially

2.2.2 Reshaping

- We can use reshape method to reshape an array. Consider an array with shape (a1, a2, a3, . . . , aN). We can reshape and convert it into another array with shape (b1, b2, b3, . . . , bM). The only required condition is: a1 x a2 x a3 . . . x aN = b1 x b2 x b3 . . . x bM . (i.e original size of array remains unchanged.)

2.2.3 Flattening

- We can use flatten method to get a copy of array collapsed into one dimension. It accepts order argument. Default value is 'C' (for row-major order). Use 'F' for column major order.

```
>>> data.reshape(2, 3)
array([[1, 2, 3],
       [4, 5, 6]])
>>> data.reshape(3, 2)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

Figure 6. reshaping array

You can index and slice NumPy arrays in the same ways you can slice Python lists.

```
>>> data = np.array([1, 2, 3])

>>> data[1]
2
>>> data[0:2]
array([1, 2])
>>> data[1:]
array([2, 3])
>>> data[-2:]
array([2, 3])
```

Figure 7. Indexing

2.3 Indexing and Slicing

Refer Figure:6 where data is a 3*2 matice containing integers from 1 to 6 in order.

2.4 Datatypes

- Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that we can use to construct arrays. Example:

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

```
dtype('int32')
```

Figure 8. Datatype

2.5 Array Math

2.5.1 Operations on single array

- We can use overloaded arithmetic operators to do elementwise operation on array to create a new array. In case of +=, -=, *= operators, the exsisting array is modified.

2.5.2 Unary Operators

- Many unary operations are provided as a method of ndarray class. This includes sum, min, max, etc. These functions can also be applied row-wise or column-wise by setting an axis parameter.

2.5.3 Binary Operators

- These operations apply on array elementwise and a new array is created. You can use all basic arithmetic operators like +, -, /, , etc. In case of +=, -=, = operators, the exsisting array is modified.

2.5.4 Universal Functions

The aforementioned operations can also be done using ufuncs like np.add, np.subtract, np.multiply, np.divide, np.sum, etc.]NumPy provides familiar mathematical functions such as sin, cos, exp, etc. These functions also operate elementwise on an array, producing an array as output.

The aforementioned operations can also be done using ufuncs like np.add, np.subtract, np.multiply, np.divide, np.sum, etc.

2.6 Broadcasting

- Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array. Numpy broadcasting allows us to perform this computation and helps making the code concise and faster.

3. Pandas

3.1 Panda Library

- Dealing with dataframes? 'Pandas' comes to your rescue. It is fast, flexible and very powerful as well as easy to use open source data analysis and manipulation tool, built on top of Python programming language.

```
import pandas as pd
```

Figure 9. Importing Pandas

3.2 Why Pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories. it can clean messy data sets, and make them readable and relevant, and relevant data is very important in data science. Unlike NumPy library which provides objects for multi-dimensional arrays, Pandas provides in-memory 2d table object called Dataframe. It is like a spreadsheet with column names and row labels.

3.3 Key Features

1. Fast and efficient DataFrame object with default and customized indexing.
2. Data alignment and integrated handling of missing data.
3. Group by data for aggregation and transformations.
4. High performance merging and joining of data.
5. Time Series functionality.

3.4 Importing file as dataframe :

```
df = pd.read_csv('Astrosat_Catalog.csv')
```

	ra	dec	Source_Name	SIMBAD_Name	Final_Type	Astrosat_Flag
0	8.233750	-73.805278	J0032.9-7348	RX J0032.9-7348	HMXB	0
1	11.210000	33.021389	0042+323	4U 0042+32	LMXB	0
2	12.260417	-72.847778	J0049-729	RX J0049.0-7250	HMXB	0
3	12.373333	-73.182222	J0049-732	[MA93] 300	HMXB	0
4	12.686250	-73.268056	J0050.7-7316	V* DZ Tuc	HMXB	0

Figure 10. CSV in Jupyter notebook

3.5 Operations on an imported CSV file :

```
df[['ra','dec','Source_Name']]    #Select multiple columns with specific names
df.head(n) or df.tail(n)          #Select first n rows/columns
df.iloc[10:20]                    #Select rows by position
df.loc[df['ra']>150,['ra','dec']] #Select columns of 'ra' & 'dec' with value of ra>150
df.insert(7,'dec1',df['dec']*np.pi/180) #Creating new column at index 7, name 'dec1', value as mentioned.
```

Figure 11. Some common functions provided by Pandas

3.6 Creating a Dataframe and exporting it to CSV format :

```
Data = []
for i in range(1,111):
    messier = []
    Messier = str("Messier ") + str(i)
    messier.append(Messier)
    coordinates = SkyCoord.from_name(Messier)
    messier.append(coordinates.ra.degree)
    messier.append(coordinates.dec.degree)
    Data.append(messier)
df = pd.DataFrame(Data, columns=['Messiers','RA','DEC'])
df.to_csv("Messiers.csv",index=None)
```

Figure 12. Creating a list of Messiers and their coordinates using Astropy library functions and storing it in form of Dataframe using Pandas and saving it as CSV file

3.7 Basic functionalities offered by Pandas :

- **size:** Returns the number of elements in the underlying data.
- **values:** Returns the series as *ndarray*.
- **T:** Transposes rows and columns.
- **dtypes:** Returns the data type.
- **ndim:** Returns the number of dimensions of underlying data.
- **shape:** Returns a tuple representing the dimensionality of the DataFrame.

4. Matplotlib

4.1 An Introduction to Matplotlib

Matplotlib is a library in Python that offers functions to plot data. More specifically, we use the matplotlib.pyplot library.

4.2 Importing matplotlib.pyplot

This is how we import the library into the Python environment: Figure 13 The plot() function The library offers a

```
In [1]: import matplotlib.pyplot as plt
```

Figure 13. Importing matlplotlib.pyplot.

plot() function that can be used to plot 2 sets of data against each other. The syntax for the plot() function is as follows: plt.plot(x_data,y_data) where x_data and y_data are the sets(may be numpy arrays, pandas dataframes or lists) for the data along x and y axes respectively. There are further attributes to the plot() function, a few of which include:

- Labels of the axes
- Color of the plot
- Type of line used (solid, dashed, etc...)
- Title of the plot
- Legends

. However, note that plot() is used to plot a continuous graph. To plot discrete points, matplotlib.pyplot offers the scatter() function which is very similar to the plot() function in the way that it takes parameters as well as attributes. The plot() function also offers many types of plots such as Mollweide projections, a very common projection used in astronomy as well as cartography.An implementation of the Mollweide projection is attached in Figure 14.

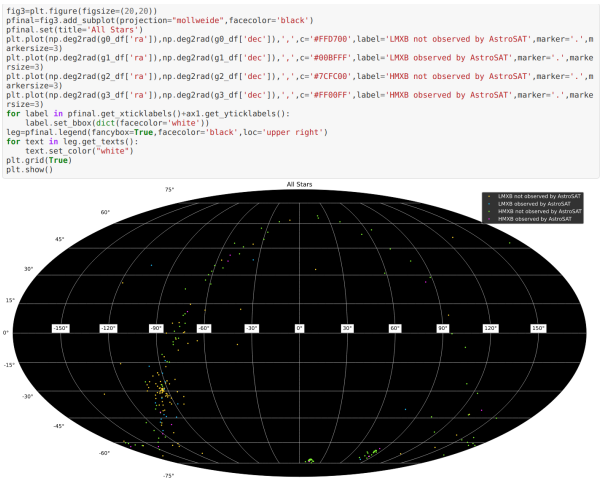


Figure 14. A Mollweide porjection classifying LMXB (low mass X-ray binaries) and HMXB (high mass X-ray binaries) based on whether or not they were detected by AstroSAT

4.3 Subplots

The subplot() utility of matplotplib allows us to create multiple subplots in a single figure. An example of subplots showing

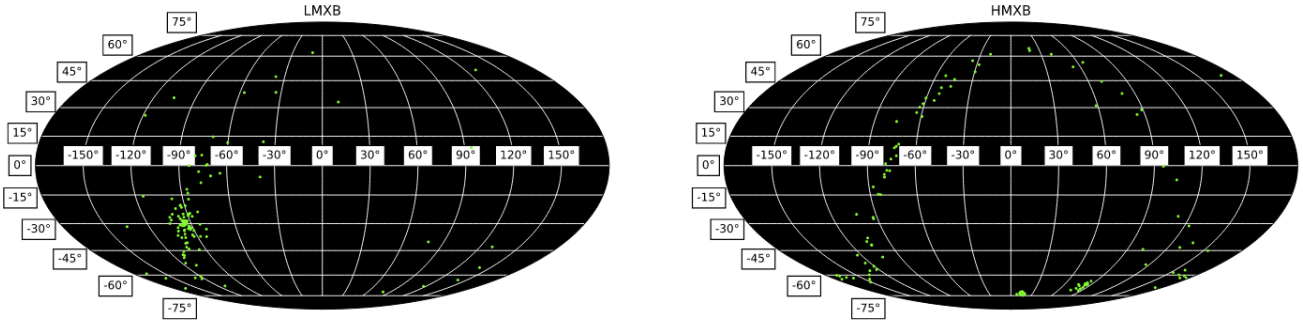


Figure 15. HMXB and LMXB sources.

LMXB and HMXB is attached in Figure 15. Note the distinct arc in the subplot for HMXB sources, this is the Milky Galaxy observed in the Mollweide Projection.

4.4 Applications of matplotlib in astronomy

We can use these techniques to plot constellations as well as plot and analyze data gathered by several sources.

Attached in Figure 16 is a plot of strain data versus time for a black-hole binary. Figure 17 shows the constellation

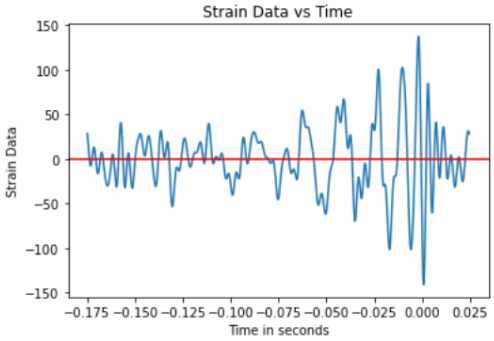


Figure 16. Plot of Strain data vs Time.

Orion plotted in a stereographic projection using matplotlib. The size of each star is determined using its measured intensity (stellar magnitude).

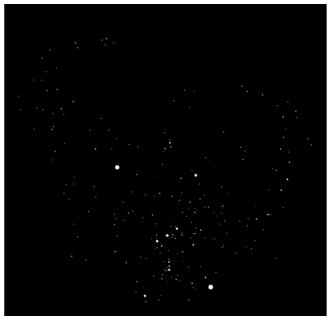


Figure 17. Orion

Week 2 : Scipy and Astropy

Abstract
Astropy is a collection of software packages written in the Python programming language and designed for use in astronomy. SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, FFT, signal and image processing, ODE etc.

[1] *Material https://github.com/astroclubiitk/computational-astrophysics*

Contents

1. Astropy coordinates

In general when we want to describe a location in 3D-space,we use coordinates of any system(Polar,Cylindrical,etc) to define it.When we want to tell our location on earth, we generally use "Geographic Coordinate System(GCS)"which includes longitudes and latitudes.The GPS which we use uses the "World Geodetic System(WGS)".

Similarly to describe a location in space, astronomers use various coordinate systems. And AstroPy reduces the work by containing various coordinate systems used in astronomy under package "astopy.coordinates".

In this project we come across various coordinate systems. Some of them are:

International Celestial Reference System(ICRS)

World Coordinate System(WCS) and FITS WCS

Fifth Fundamental Catalogue(FK5)

Galactic Coordinate System

Ecliptic Coordinate System

1.1 Co-Ordinate Systems

Now, let us see about some coordinate systems used in astronomy:

1.1.1 ICRS

The International Celestial Reference System (ICRS) is the current standard celestial reference system adopted by the International Astronomical Union (IAU). Its origin is at the barycenter of the Solar System, with axes that are intended to be "fixed" with respect to space.ICRS coordinates are approximately the same as equatorial coordinates.

In Equitorial Coordinates we describe a location by Right Ascension(ra) and Declination(dec) which are similar to longitudes and latitudes.

1.1.2 WCS and FITS WCS:

The World Coordinate System (WCS) is a set of transformations that map pixel locations in an image to their real-world units, such as their position on the sky sphere. These transformations can work both forward (from pixel to sky) and backward (from sky to pixel).World Coordinate System (WCS) is a set of transformations that map pixel locations in an image to their real-world units, such as their position on the sky sphere. These transformations can work both forward (from pixel to sky) and backward (from sky to pixel).

Flexible Image Transport System (FITS) is a digital file format useful for storage, transmission and processing of

scientific and other images. It is the defacto standard used by many sky tessellation softwares.

1.1.3 FK5

The FK5 is part of the “Catalogue of Fundamental Stars” which provides a series of six astrometric catalogues of high precision positional data for a small selection of stars to define a celestial reference frame. J2000 refers to the instant of 12 PM (midday) on 1st January, 2000. FK5 was published in 1991 and added 3,117 new stars.

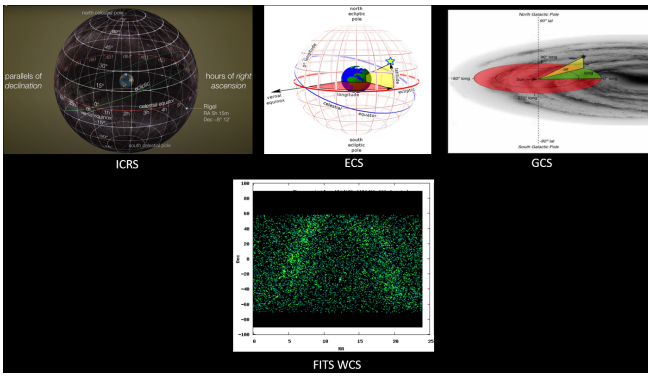
1.1.4 GCS

The Galactic coordinate system is a celestial coordinate system in spherical coordinates, with its origin at the Sun, the primary direction aligned with the approximate center of the Milky Way galaxy, and the fundamental plane parallel to an approximation of the galactic plane but offset to its north. GCS has its own Galactic longitude and Galactic latitude.

1.1.5 ECS

The Ecliptic coordinate system commonly used for representing the positions and orbits of Solar System objects. The system’s origin can either be the center of the Sun or the center of the Earth, its primary direction is towards the vernal (northbound) equinox, and it follows a right-handed convention.

The Vernal equinox, two moments in the year when the Sun is exactly above the Equator and day and night are of equal length; also, either of the two points in the sky where the ecliptic (the Sun’s annual pathway) and the celestial equator intersect.



1.2 Using Astropy

1.2.1 Basic usage

We import the coordinate systems package by

”*astropy.coordinates*”

Then we use function ”SkyCoord” to assign the coordinates to various systems.

Then we can obtain the details of the coordinates in various units and obtain it in string by ”to-string()” method as shown in example,

```
In [6]: import numpy as np
        import pandas as pd
        from astropy import units as u
        from astropy.coordinates import SkyCoord
        c = SkyCoord(ra=88.79293899*u.degree, dec=7.407064*u.degree, frame='icrs'
        print(c)

        <SkyCoord (ICRS): (ra, dec) in deg
            (88.79293899, 7.407064)>

In [5]: c.ra.degree

Out[5]: 88.79293899

In [7]: c.ra.hour

Out[7]: 5.9195292660000005

In [8]: c.ra.hms

Out[8]: hms_tuple(h=5.0, m=55.0, s=10.305357600001628)

In [12]: c.ra.radian

Out[12]: 1.5497291378979483

In [11]: print(c.to_string('dms'))

         88d47m34.5804s 7d24m25.4304s

In [13]: print(c.to_string('decimal'))

         88.7929 7.40706
```

1.2.2 Transformations

We can also transform the coordinates from one coordinate system to another as shown in example, So in the last part of last example, we can see that we have imported ”J1975”. It corresponds to the reference equinox at epoch J1975. Epoch refers to a particular point of time.

```
In [14]: c_icrs = SkyCoord(ra=10.68458*u.degree, dec=41.26917*u.degree, frame='icrs')
         c_icrs.galactic

Out[14]: <SkyCoord (Galactic): (l, b) in deg
             (121.17424181, -21.57288557)>

In [15]: c_fk5 = c_icrs.transform_to('fk5')  # c_icrs.fk5 does the same thing
         c_fk5

Out[15]: <SkyCoord (FK5: equinox=J2000.000): (ra, dec) in deg
             (10.68459154, 41.26917146)>

In [17]: from astropy.coordinates import FK5
         c_fk5.transform_to(FK5(equinox='J1975'))

Out[17]: <SkyCoord (FK5: equinox=J1975.000): (ra, dec) in deg
             (10.34209135, 41.13232112)>
```

2. Scipy - Roots

SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering. In our project, we use SciPy to do various tasks like :

Solving Equations(Roots)

Curve Fitting

Solving Ordinary Differential Equations

2.1 Solving equations using SciPy

We use the ”scipy.optimise” package with a function ”root” to solve the equations. The following piece of code is from ”scipy.optimise import root”

```
from scipy.optimise import root
import numpy as np
```

For instance, to obtain the roots of linear equation,

$$4x-5y+8=0, 3x+2y-17=0$$

We solve it as follows

```
from scipy.optimise import root
import numpy as np
def f(x):
    return [4*x[0]-5*x[1]+8, 3*x[0]+2*x[1]-17]
val=root(f,[0,0])
print(val.x) #returns the array x, ie [x,y] of the equations
```

Another function including trigonometric identities,

$$x-3\sin(x)=0$$

```
from scipy.optimise import root
import numpy as np
def f(x):
    return x-3*np.sin(x)

#The equation will have 3 roots so 3 initial guesses need to be provided
val=root(f,[-2,0,3])
print(val.x)
```

3. Ordinary differential equations

SciPy stands for scientific python. It is a scientific computation library that provides more utility functions for optimization, statistics and signal processing.

3.1 Differential equation solver

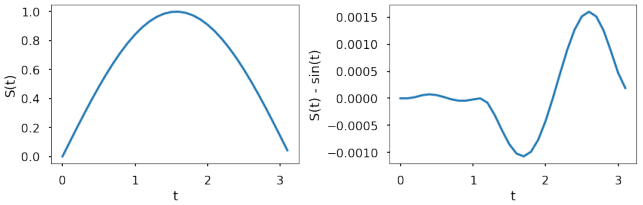
Differential equations are solved in Python with the Scipy.integrate package using function solve_ivp. We use python function as follows: from scipy.integrate import solve_ivp

3.2 Solving linear differential equations

An example of using scipy.integrate is with the following differential Use solveivp to approximate the solution to this initial value problem over the interval [0,π]. Plot the approximate solution versus the exact solution and the relative error over time.

$$\frac{dS(t)}{dt}=cos(t)$$

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import solve_ivp

plt.style.use('seaborn-poster')

%matplotlib inline

F = lambda t, s: np.cos(t)

t_eval = np.arange(0, np.pi, 0.1)
sol = solve_ivp(F, [0, np.pi], [0], t_eval=t_eval)

plt.figure(figsize = (12, 4))
plt.subplot(121)
plt.plot(sol.t, sol.y[0])
plt.xlabel('t')
plt.ylabel('S(t)')
plt.subplot(122)
plt.plot(sol.t, sol.y[0] - np.sin(sol.t))
plt.xlabel('t')
plt.ylabel('S(t) - sin(t)')
plt.tight_layout()
plt.show()
```

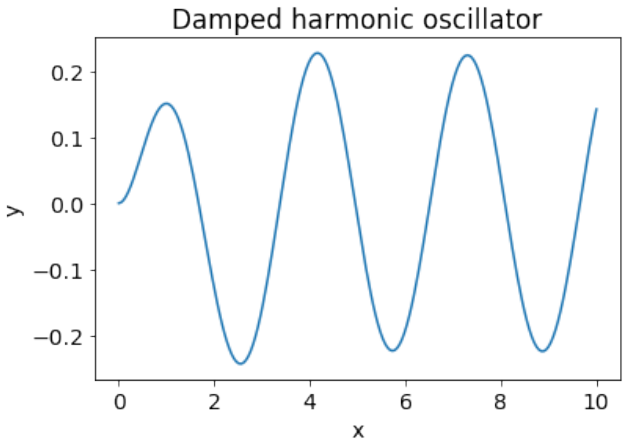


3.3 Higher order differential equations

Suppose we have a second-order ODE such as a damped simple harmonic motion equation, $y''+2y'+2y=cos(2x)$ $y(0)=0, y'(0)=0$

We can turn this into two first-order equations by defining a new dependent variable. For example, $zyz+2z+2y=cos(2x)$ $z(0)=y(0)=0$ We can solve this system of ODEs using as follows:

```
def dU_dx(U, x):
    # Here U is a vector such that y=U[0] and z=U[1]. This function should return [y', z']
    return [U[1], -2*U[1] - 2*U[0] + np.cos(2*x)]
U0 = [0, 0]
xs = np.linspace(0, 10, 200)
Us = odeint(dU_dx, U0, xs)
ys = Us[:,0]

plt.xlabel("x")
plt.ylabel("y")
plt.title("Damped harmonic oscillator")
plt.plot(xs,ys);
```



4. Units and constants

AstroPy is a collection of packages written in the Python and designed for use in astronomy. AstroPy contains numerous constants, units, coordinate systems and other functions that are helpful for computing Astronomical Data.

4.1 Constants

- AstroPy contains various constants which are helpful in astronomy in ”astropy.constants” package. It also contains the additional meta-data describing their history of origin, references and uncertainties. We access constants by using ”from astropy import constants as const” line.

- For instance let us find about ”Speed of Light(c)” constant:

```
In [1]: from astropy import constants as const
        print(const.c)
```

```
  Name   = Speed of light in vacuum
  Value  = 299792458.0
  Uncertainty  = 0.0
  Unit  = m / s
  Reference = CODATA 2018
```

- We can also use the constants in different units as shown

```
In [2]: from astropy import constants as const
        print(const.c.to('km/s'))
        print(const.c.cgs)
```

```
299792.458 km / s
29979245800.0 cm / s
```

- We can also different versions of constants which may vary in their ”Precision in Value”,”Uncertainity”,etc.Physical CODATA constants are in modules with names like codata2010, codata2014, or codata2018. Astronomical constants defined by the International Astronomical Union (IAU) are collected in modules with names like iau2012 or iau2015.Here are some examples

```
In [3]: from astropy.constants import codata2014 as const14
        print(const14.h)
        print('-'*40)
        from astropy.constants import codata2018 as const18
        print(const18.h)
```

```
  Name   = Planck constant
  Value  = 6.62607004e-34
  Uncertainty  = 8.1e-42
  Unit  = J s
  Reference = CODATA 2014
----------------------------------------
  Name   = Planck constant
  Value  = 6.62607015e-34
  Uncertainty  = 0.0
  Unit  = J s
  Reference = CODATA 2018
```

```
In [4]: from astropy.constants import iau2012 as const12
        print(const12.L_sun)
        print('-'*60)
        from astropy.constants import iau2015 as const15
        print(const15.L_sun)
```

```
  Name   = Solar luminosity
  Value  = 3.846e+26
  Uncertainty  = 5e+22
  Unit  = W
  Reference = Allen's Astrophysical Quantities 4th Ed.
------------------------------------------------------------
  Name   = Nominal solar luminosity
  Value  = 3.828e+26
  Uncertainty  = 0.0
  Unit  = W
  Reference = IAU 2015 Resolution B 3
```

4.2 Units

- 1. While trying to obtain some astronomical values,we may encounter some situations where we require units(like for conversions,rechecks,etc).So AstroPy contains ”Units” package which makes the work easy.

- 2. We access to units by using ”from astropy import units as u”.We generally assign a unit to a quantity by multiplying ”u.unit”.Here the unit can be of any type.

- 3. We can also change the value of quantity in different units and systems.

- 4. A special type of unit in the astropy.units package is the dimensionless quantity.This can either be initialised directly or come out from units cancelling out.

- 5. It also has access to some constant values like ”Mass of Sun”,”Radius of Earth”,etc.

- 6. Astropy can also perform ’implicit’ conversions,like converting wavelength of light to frequency (in vacuum), by specifying the equivalencies parameter.”spectral()” is a function that returns an equivalency list to handle conversions between wavelength, frequency, energy, and wave number

The following points are shown below with example

```
In [10]: import numpy as np
         from astropy import units as u
         distance2 = 45 * u.m
         distance1 = 5*u.m
         time = 2 * u.s
         speed = (distance2 - distance1)/time
         print(speed)
         print(speed.to(u.km/u.hour))
         print(speed.value)
         print(speed.unit)
         print((1*u.Joule).cgs)
         print((10*u.cm).si)
         refractive_index = 1.5 *u.dimensionless_unscaled
         print(refractive_index.value,refractive_index.unit)
         distance_ratio1 = (1*u.m)/(2*u.m)
         print(distance_ratio1.value,distance_ratio1.unit)
         print(distance_ratio1.unit == refractive_index.unit)
         AvgDensity_of_sun = 1*u.Msun/(4*np.pi/3*(1*u.Rsun)**3)
         print(AvgDensity_of_sun.to(u.kg/u.m**3))
         print((1000 * u.nm).to(u.Hz, equivalencies=u.spectral()))

         20.0 m / s
         72.0 km / h
         20.0
         m / s
         10000000.0 erg
         0.1 m
         1.5
         0.5
         True
         1409.7798243075256 kg / m3
         299792457999999.94 Hz
```

5. Cross matching

5.1 Introduction

In Astronomy large data sets and catalogs are present ,mapping the different regions of the universe from various observatories on earth. Some datafields are:

- Right Ascension
- Declination
- Magnitude
- Parallax
- Luminosity
- Radius
- Temperature

To determine that two observations are pointing to the same source,we use cross-matching.

5.2 Distance based algorithm

Our project uses the very popular distance based algorithm where we calculate the on sky distances of two given RA(Right Ascension) and DEC(Declination) using the Haversine Formula and comparing it to a maximum error(around 10-20 arcsecs) The Haversine formula:-

$$d = 2r\arcsin(\sqrt{hav(\varphi_2-\varphi_1)+\cos(\varphi_2)\cos(\varphi_1)hav(\lambda_2-\lambda_1)}$$

where,

- φ_i is the latitude of point i in rad
- λ_i is the longitude of point i in rad
- $hav(\theta) = \sin^2(\theta/2)$

5.3 AstroPy Support

AstroPy easily packages these algorithms into this convenient function:-

```
(c1).match_to_catalog_sky(c2)
```

Here c1 and c2 are simply coordinate pairs for the different catalogs we consider.These are the RA and DEC values in degrees and stored as arrays. The output of the function is a tuple packed out into three values.

1. idx : integer array : The index of the nearest source in catalog 2 for each source in catalog 1.
2. d2d : angle : The on-sky(angular)(arcsec) for each element in the two catalogs
3. d3d : Quantity : The 3D separation between each element

5.3.1 Conclusion

Application of the above methods to various datasets ,gives us the indices which satisfy the given error margin. We then safely declare that the objects at those two indices are actually the same. If we recognise an object that does not give a true cross-match with any source present in the vast catalogs compiled,one can safely state that the object must indeed be a new discovery.

6. Curve Fitting

6.1 What is Curve Fitting

Curve fitting, as the term suggests, finding a mathematical function that best describes a set of data points, usually observed. As observation is prone to some errors, curve fitting is usually not so easy, because of *noise* in the data points. Practically, Curve-fitting is just an optimisation problem, of finding the best representative to a collection of observations. For

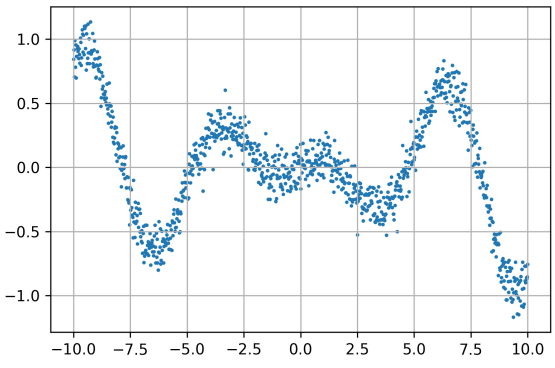


Figure 1. Possible set of Observations

further discussion, let's be limited to two dimensions, which is easier to visualise. The x-axis is the independent variable or

the input to the function. The y-axis is the dependent variable or the output of the function. We don't know the form of the function that maps examples of inputs to outputs, but we suspect that we can approximate the function with a standard function form.
Curve fitting involves first defining the functional form of the mapping function (also called the basis function), then searching for the parameters to the function that result in the minimum error.
Error is calculated by using the observations from the domain and passing the inputs to our candidate mapping function and calculating the output, then comparing the calculated output to the observed output.

6.2 Curve Fitting Algorithms

Scipy itself uses non-linear squares method, but let's start from a bit basic.
The key to curve fitting is to form a mapping function at start. Let's consider a linear mapping function.

$$y = mx + c$$

The coefficients are parameters that are to be adjusted to find the best fit, using an optimisation algorithm, which is called linear regression. So far, linear equations of this type can be fit by minimizing least squares that is, finding a cost function like:

$$J = \frac{\sum_1^n (y - y_{original})^2}{n}$$

and reducing this cost function through gradient descent like:

$$\delta = \nabla J$$

$$\delta_m = Jm = \frac{\sum_{i=1}^n x_i (y_i - y_{original,i})}{n}$$

$$\delta_c = Jc = \frac{\sum_{i=1}^n (y_i - y_{original,i})}{n}$$

and can be calculated analytically. This means we can find the optimal values of the parameters using a little linear algebra.

$$m_{new} = m - \alpha \cdot \delta_m$$

$$c_{new} = c - \alpha \cdot \delta_c$$

Where α is the length of jump along the gradient vector, and that matters.

The equation does not have to be a straight line.
We can add curves in the mapping function by adding exponents. For example, we can add a squared version of the input weighted by another parameter:

$$y = a_1 x^2 + a_2 x + a_3$$

Similar reasoning as above would work in such cases, even after introducing non-linear terms.

6.3 Scipy's implementation of Curve Fitting

Now moving on to how Scipy implements Curve fitting through its 'curve_fit()' function, The function takes the same input and output data as arguments, as well as the name of the mapping function to use (using linear model in following code example):

```
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
def f(x, m, c):
    return m*x+c
xdata = np.linspace(0, 5, 500)
ydata = f(xdata, 2, 1) +
0.25*np.random.randn(len(xdata))
p_opt, p_cov = curve_fit(f, xdata, ydata)
print(p_opt)
```

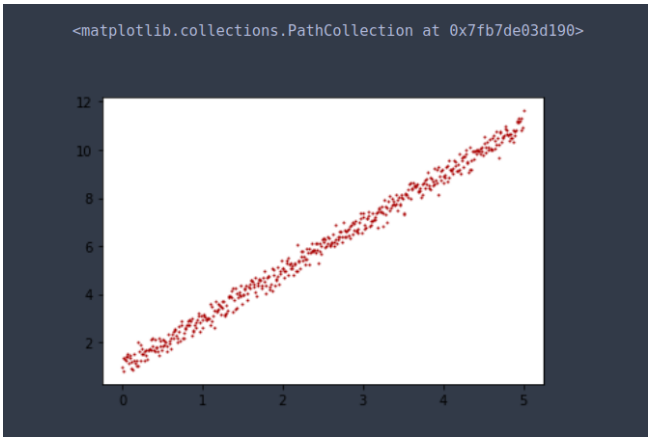


Figure 2. This is the plot with 'created' observations.

Output is [2.01640064 0.95691782]
Clearly, this is quite close to the original value of [2,1]. 'p_cov' is the covariance matrix generated along with optimum parameters.

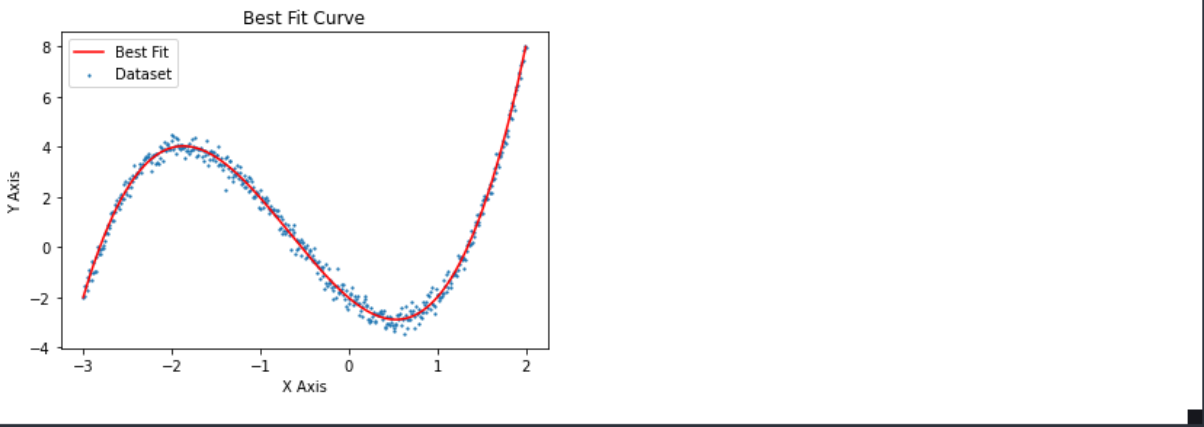
This idea can, as explained earlier, be easily extended to non-linear curves as well, where our base function is set as powers of x or transcendental function of x, and curve_fit() would find optimum parameters to fit given observation set to such a function.

Another application example we did was for polynomial curve fitting:

```python
#Define the model
def f(x,a,b,c,d):
    return a*x**3+b*x**2+c*x+d

#Let's generate some dataset
xdata=np.linspace(-3,2,500)
ydata=f(xdata,1,2,-3,-2)+0.25*np.random.randn(len(xdata)) #We've given (a,b,c,d)=(1,2,-3,-2)

#Obtain the best fit parameters
p_opt, p_cov = cf(f,xdata,ydata)
print(p_opt)

#Plot the best fit curve
plt.scatter(xdata,ydata,s=1.5,label='Dataset')
plt.plot(xdata,f(xdata,*p_opt),'r',label='Best Fit')
plt.xlabel("X Axis")
plt.ylabel("Y Axis")
plt.title('Best Fit Curve')
plt.legend()
plt.show()
```

[10] ✓ 0.1s Python

```
[ 0.99670274  1.99831723 -2.98112367 -2.00815642]
```



+ Code + Markdown

Week 3 : Fourier Transform and Cepheid Variables

Abstract
In mathematics, a Fourier transform is a mathematical transform that decomposes functions depending on space or time into functions depending on spatial or temporal frequency. It is one of the seemingly ubiquitous fields of maths, physics domains. A drawback of the Fourier transformation techniques is that it can only be used for evenly spaced data. In Astronomy, several times it happens that the period is long enough that uniform observations cannot be made even within the period time. In such a case we have to resort to Periodograms method which are estimations of the power spectrum of Fourier transforms. In this article we discuss Lomb-Scargle periodogram. Cepheid variable is a type of star that pulsates radially, varying in both diameter and temperature and producing changes in brightness with a well-defined stable period and amplitude. They act as cosmic benchmarks because of a strong direct relationship between their luminosity and pulsation period, which helps astronomers measure distances.

Contents

1. Fourier Transform

In mathematics, a Fourier transform (FT) is a mathematical transform that decomposes functions depending on space or time into functions depending on spatial or temporal frequency. To decipher the crux of Fourier transformation and its immense contribution in the field of astronomy and astrophysics, let us bring in an example of a pallet of paints of various colours mixed, making such a mess is relatively easy compared to the stigma of separating this, and what if we replace the problem of this physical entity with sounds, frequencies and many more phenomena. The FT primarily deals with the decomposition of frequencies from an added up non -sinusoidal wave and is one of the seemingly ubiquitous fields of maths, physics domains. Of course, there can be a combination of pure sinusoidal waves. Still, all these account for simple addition, and the resultant function is no more sinusoidal. Dealing with such a function and decomposing the elements is sure to be a herculean task until we have the FT!

There is an explicit and intriguing phenomenon happening in the so-called winding machine mechanism of the FT (not exactly FT, but metaphorically, they perform the same function)

The task is to create a so-called winding machine that could differentiate out the original pure sinusoidal frequencies that added up to one messy non-sinusoidal function. The working of FT is similar to the lines followed. We now have

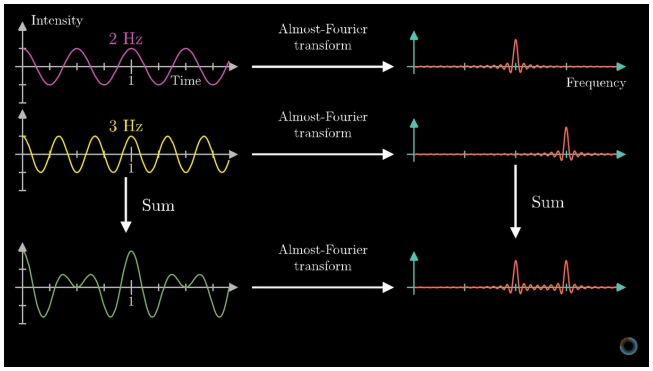


Figure 1. Frequencies and their respective FT

a graph of the non-sinusoidal function. We are now trying to plot it in a 2D plane, explicitly stated in a circle, which would be like assuming to remove the function from the function vs time graph and marking the same graph in a circle (2D) arena where this would seemingly be felt like a polar kind of representation of the function, it would be fascinating to observe that at some stage the polar graph made out of the standard non-sinusoidal function enwraps itself to a particular configuration towards to positive x-axis (in case the function is positive) especially when the frequency of the first graph matches with the second.

To explain it further, one can imagine the wound up circular or polar form of the graph to have a centre of mass, and as soon as the function changes with the change in time, the centre of mass wobbles around a bit, and for most of the winding frequencies(particular frequencies where the second graph takes a concrete shape) the peaks and valleys (compared in the original frequency vs time function) are all spaced out around the circle in such a way that the centre of mass stays pretty close to the origin. When the winding frequency is

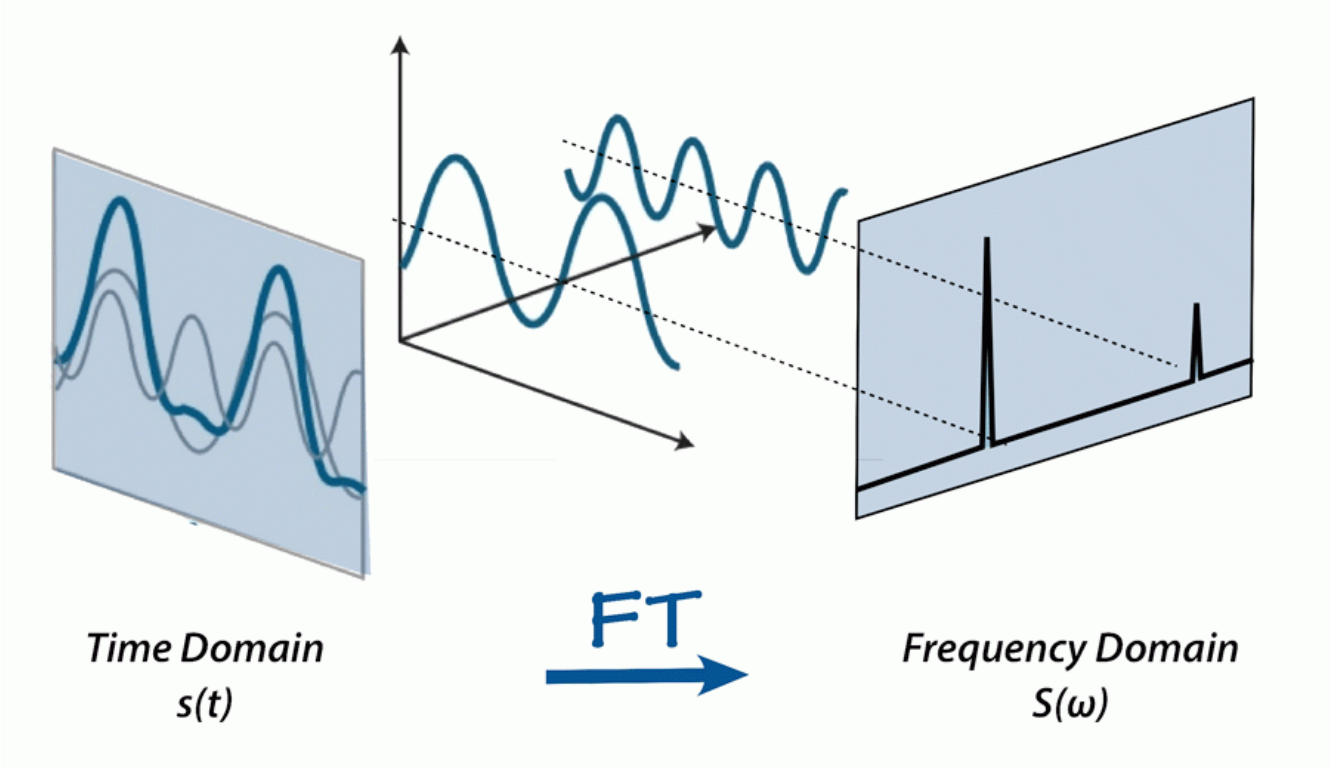


Figure 2. Source:https://aavos.eu/glossary/fourier-transform/

the same as the frequency of the given signal, all the peaks align to the right and valleys to the left. This (centre of mass), when plotted, provides an exact curve where there is a clear-cut spike whenever the winding frequency matches with the frequency of the given signal. These places indeed deliver the values of the frequencies that made up the original signal, so this is one hell of an example where we ensured a winding machine-like mechanism that could hopefully decipher or decompose the frequencies that make up a given frequency. But is this necessarily or exactly the FT? not complete until we get in-depth to the mathematical aspects of the problem we have encountered.

$$\frac{1}{N}\sum_{k=1}^{n} g(t_k)\, e^{-2\pi i f t} \tag{1}$$

This mathematical expression does precisely what we wanted. It pulls out original frequencies from the jumbled-up sums (unmixing the pallet of paint). The best part is that there is another reverse procedure of what we have done, the inverse Fourier transformation, that could possibly approximate the reverse of the procedures we have followed till now. The Fourier transformation can be essentially helpful if you have a sound recording and want to pull out an unwanted pitch from the recording. All we have to do is plot the graph. At characteristic sites, we can observe a spike in the peak of the final COM graph that specifies the individual frequencies. Then, we can track the frequency related to the shrill pitch and effectively dump it from the recordings. Finally, we follow the inverse Fourier transformation to put back the pieces of the puzzle.

Consider the equation

$$\hat{g}(f) = \int_{-\infty}^{\infty} g(t)\, e^{-2\pi i f t} dt \tag{2}$$

The Euler's number is coming in handy in here. The 2π denotes the second graph that would possibly be along a circle, the frequency f is to indicate how many times the vector form the centre to the function drawn in the near-circular region (synchronised with the actual x-y plot of the given signal) turn in each time-period. Thus, the rotating vector in the area fluctuates up and down in the original signal plotted in the x-y graph and fluctuates in a pattern of a flower blooming and then shrinking in the latter graph.

$$\frac{1}{N}\sum_{k=1}^{n} g(t_k)\, e^{-2\pi i f t} \tag{3}$$

This small expression is a super elegant way to encapsulate the whole idea of winding a graph around a circle with a variable frequency fff. Now it's time to understand the COM tracking associated with deciphering the frequencies that made up the messy non-sinusoidal function. Approximate it, at least. First, we need to pick up a bunch of points from the original signal, see where the points end up on the wound up graph and taking the average.

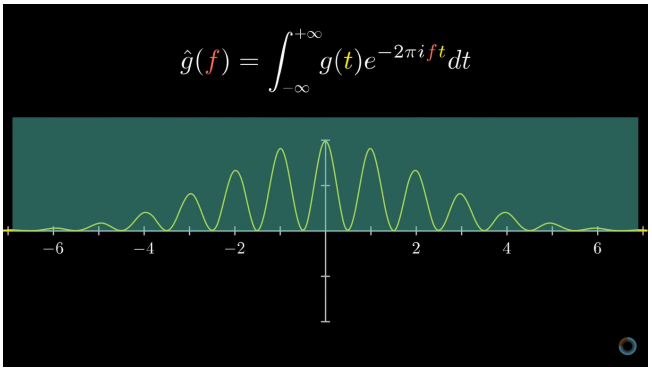


Figure 3. FT in Integral form with all possible finite time intervals

$$\hat{g}(f) = \int_{-\infty}^{\infty} g(t)\, e^{-2\pi i f t} dt \tag{4}$$

Add them together as complex numbers and divide them by the number of points you have sampled. The more points you take, the closer the graph gets and the easier it gets to solve the riddle. It can be expressed as an integral over a complex function that sure is easy to behold, but solving it is a legit nightmare. This is not necessary, but here comes the actual honest to goodness Fourier transformation. This is the same expression except for the fact that we do not divide the integral by the time interval. This implies that the given function or the distance of the COM from the origin in the winding curve is gradually a multiple of the time taken as now the division is omitted. Hence, FT of intensity versus time function is a new function that does not have time as an input but takes up frequency (winding frequency). In terms of notation, the standard convention to call this function is $\hat{g}$; now, this function's output is a complex number. In a 2D plane, some point corresponds to the strength of a given frequency in the original signal. The plot that has been graphed for the FT is just the fundamental component of the output, the x coordinate separately, daunted in the expression.

Often the limits are taken from $-\infty \to \infty$. There is a conventional FT by the name discrete FT or famously the "Schuster" periodogram. But we are more likely to give a heads up for the Lomb Scale periodogram.

2. Lomb-Scargle Periodogram

A drawback of the Fourier transformation techniques is that it can only be used for evenly spaced data. In Astronomy, several times it happens that the period is long enough that uniform observations cannot be made even within the period time. In such a case we have to resort to other methods. One such method is the use of Periodograms which are estimations of the power spectrum of Fourier transforms. The periodogram that we will discuss is the Lomb-Scargle periodogram. The Lomb–Scargle periodogram is a well-known algorithm for detecting and characterizing periodicity in unevenly sampled time-series and has seen particularly wide use within the astronomy community. As an example of a typical application of this method, consider the data shown in Figure 1: this is an irregularly sampled time-series showing a Cepheid variable (OGLE-LMC-CEP-0005), with unfiltered intensity magnitude. (We will discuss more about what are Cepheid variables in the next Section.) By eye, it is clear

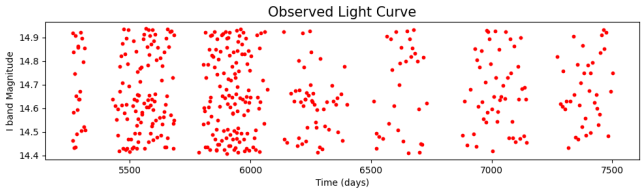


Figure 4. Uneven Data

that the brightness of the object varies in time with a range spanning approximately 0.171 mag, but what is not immediately clear is that this variation is periodic in time. The Lomb–Scargle periodogram is a method that allows efficient computation of a Fourier-like power spectrum estimator from such unevenly sampled data, resulting in an intuitive means of determining the period of oscillation. The Lomb–Scargle periodogram computed from these data is shown in figure 5. The Lomb–Scargle periodogram here yields an estimate of

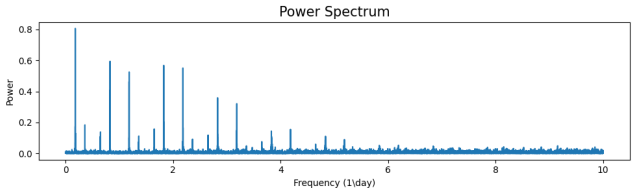


Figure 5. Lomb-Scargle Periodogram

the Fourier power as a function of period of oscillation, from which we can read off the period of oscillation of approximate 5.611 days. Figure 6 shows a folded visualization of the same data as Figure 4 i.e., plotted as a function of phase rather than time. Here we employ the method called phase-folding. Basically, we take a starting point and from there we assign each time stamp a phase value according to the time period. Like if we start from t=0 and the time period is T=2, then t=1 would be assigned the phase 0.5. Phase folding brings out the structure or shape of the periodic variation.

3. Cepheid Variables

A Cepheid variable is a type of star that pulsates radially, varying in both diameter and temperature and producing changes in brightness with a well-defined stable period and amplitude. Stating simply; they brighten and dim periodically. Cepheid variables act as cosmic benchmarks because of a strong direct relationship between their luminosity and pulsation period, which helps measuring distances a lot easier for astronomers.

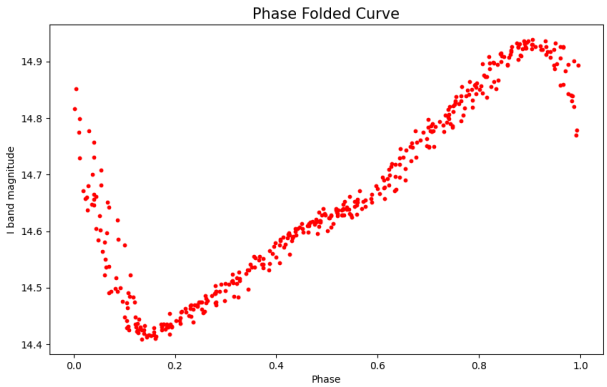


Figure 6. Phase folded Graph

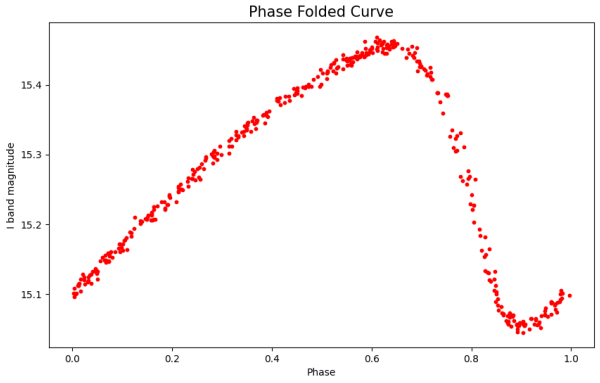


Figure 8. Phase folded Graph for OGLE-LMC-CEP-4364

Cepheid variables are reasonably abundant and very bright. Astronomers can identify them not only in our Galaxy, but in other nearby galaxies as well. If one requires the distance to a given galaxy one first locates the Cepheid variables in this galaxy. From these observations one determines the period of each of these stars, which tells us about its luminosity which then can be easily converted to distance in light years. The process to do this has already been covered in the previous section. Below are the phase folded graphs for some more Cepheid variables.

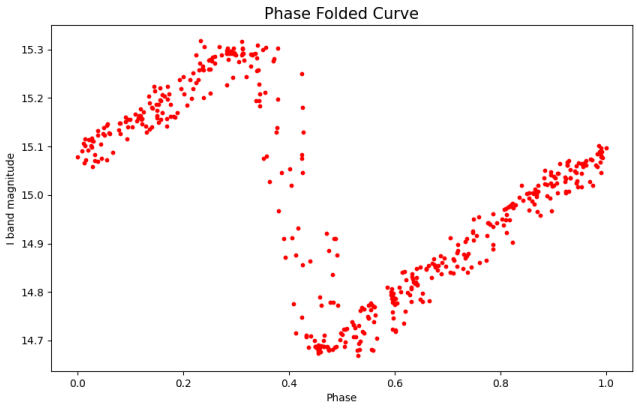


Figure 7. Phase folded Graph for OGLE-LMC-CEP-0780

4. References

1. Data for Cepheid Variables is obtained from: http://ogledb.astrouw.edu.pl/ ogle/OCVS/index.php. https://starchild.gsfc.nasa.gov/docs/StarChild/questions/cepheids.html

2. Fig 1 and Fig 3 obtained from 3Blue1Brown's *But what is the Fourier Transform? A visual introduction* https://youtu.be/spUNpyF58BY

Week 4 : Image Processing

Abstract
For astronomy and other quantitative imaging work, the Flexible Image Transport System (or FITS) format is almost universal. It includes the image data, and a header describing the data. FITS files may also be tables of data, or a cube of images in sequence. The standards developed for creating these files are slowing evolving as the needs of big data in astronomy have grown.

[1] *Material https://github.com/astroclubiitk/computational-astrophysics*

Contents

Introduction to FITS

The Flexible Image Transport System (FITS) is the most commonly used file format for astronomical data . It was initially developed by astronomers in the USA and Europe in the late 1970s to serve the interchange of data between observatories and was brought under the auspices of the International Astronomical Union in 1982. It is an open standard defining a digital file format useful for storage, transmission and processing of data: formatted as multi-dimensional arrays or tables.The FITS standard was designed specifically for astronomical data , and includes provisions such as describing photometric and spatial calibration information, together with image origin metadata.

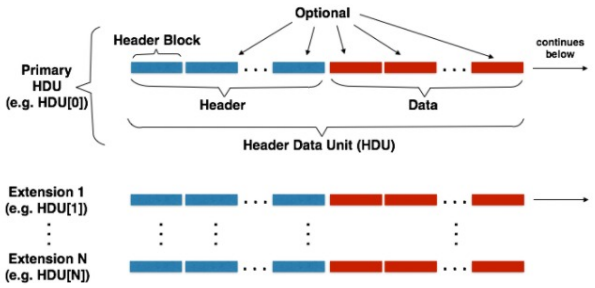


Figure 1. FITS

0.1 Why FITS?

- Storage of more bits per pixel and also floating point values.
- Storage of arbitrary number of data channels.
- No lossy compression as is typical for JPEG.
- Provides higher resolution and is capable of storing 3D data volumes.
- Support for unlimited metadata in the header, for example the sky coordinates, information about the telescope, etc. which is not provided by other file formats.
- It is backward compatible.

Softwares like SAO IMAGE DS9 can be used to open FITS file. Alternatively ,it can be done using python as well as shown below:

```
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
from astropy.utils.data import download_file
import scipy
from scipy import stats
img_file = download_file('http://data.astropy.org/tutorials/FITS-images/HorseHead.fits', cache=True )
hdu_list=fits.open(img_file)
img_data=hdu_list[0].data
plt.imshow(img_data,cmap='viridis')
plt.colorbar()
plt.show()
```

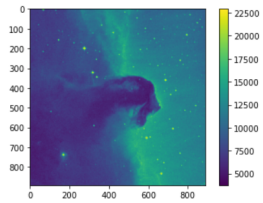
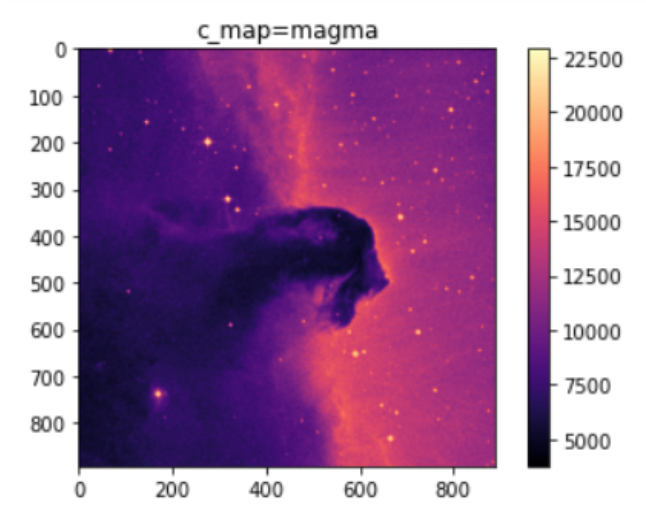


Figure 2. Code for Logarithmic Scaling

1. Image Stacking

Image stacking is a technique used to enhance astrophotographic images by reducing image noise and distortion ,thereby boosting image details.

An image stack combines a group of images with a similar frame of reference, but differences of quality or content across the set.



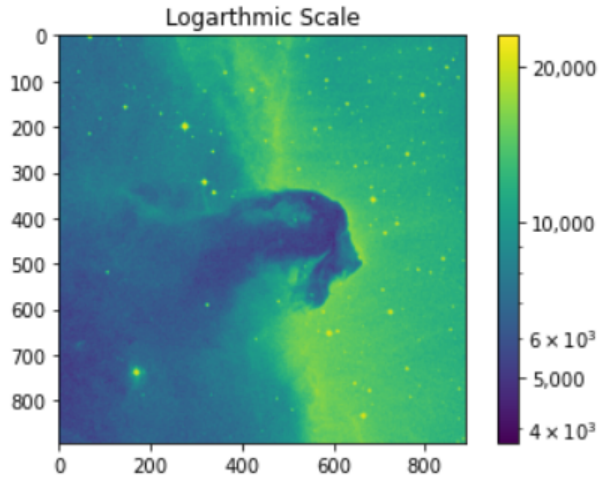


Figure 3. Logarithmic Scaling

Once combined in a stack, you can process the multiple images to produce a composite view that eliminates unwanted content or noise.

2. Bibliography

https://en.m.wikipedia.org/wiki/FITS https://photographingspace.com/how-to-use-fits/ https://astronomy.stackexchange.com/questio 15029/why-do-we-use-fits-format-for-scienti

```
base_url = 'http://data.astropy.org/tutorials/FITS-images/M13_blue_{0:04d}.fits'
image_list=[download_file(base_url.format(n),cache=True) for n in range(1,6)]
image_concat=[fits.getdata(image) for image in image_list]
final_image = np.zeros(shape=image_concat[0].shape)
for i in image_concat:
    final_image=final_image+i
histo=plt.hist(final_image.flatten(),bins='auto')
#scaling the image according to the histogram
plt.show()
plt.title('Stacked image')
plt.imshow(final_image,cmap='viridis',vmin=2.2e3,vmax=2.9e3)
plt.colorbar()
plt.show()
```

Figure 4. Code

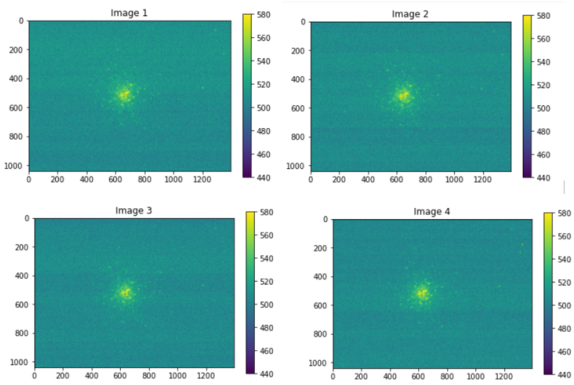


Figure 5. Image Stacking

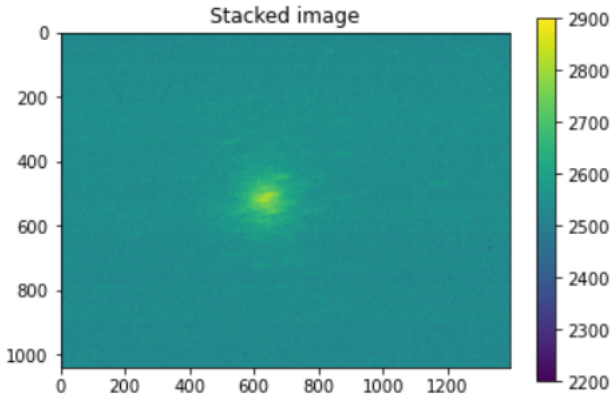


Figure 6. Stacked Image

Week 5 : Astroquery

Abstract
This week we learnt about another python library called Astroquery. Astroquery is a set of tools for querying astronomical web forms and databases. This is extremely useful in obtaining data from official catalogues and databases.

[1] *Astronomy Club, IITK*
[2] *Science and Technology Council, IITK*

Contents

1. Introduction

Astroquery is an Astropy affiliated package and is tool for accessing databases containing a star's or a system of stars' information. These tools are built on the Python requests package, which is used to make HTTP requests, and Astropy, which provides most of the data parsing functionality. For our purposes, we make use of it's packages: VizieR, SIMBAD and SDSS. VizieR gives us the most complete library of published astronomical catalogues, tables and associated data, verified and can be accessed through several interfaces. The SIMBAD astronomical database provides basic data, cross-identifications, bibliography and measurements for astronomical objects outside the solar system. While SIMBAD is a dynamic database and is updated every working day, SIMBAD is not a catalogue, and should not be used as a catalogue. The CDS also provides the VizieR database which contains published lists of objects, as well as most very large surveys. The idea now is to use both SIMBAD and VizieR as complementary research tools. Lastly, the SDSS is a project to make a map of a large part of the universe. The kinds of data include images, spectra, photometric data, and spectroscopic data.

2. Sloan Digital Sky Survey

Starting with how to use astroquery to query the SDSS, we will explain the following:

- What is SDSS?
- Querying objects and region
- Downloading Images using SDSS
- Downloading Spectra using SDSS
- Spectral templates

2.1 Introduction

The Sloan Digital Sky Survey or SDSS is a major multi-spectral imaging and spectroscopic redshift survey using a dedicated 2.5-m wide-angle optical telescope at Apache Point Observatory in New Mexico, United States. The project was named after the Alfred P. Sloan Foundation, which contributed significant funding.

SDSS uses a dedicated 2.5 m wide-angle optical telescope; from 1998 to 2009 it observed in both imaging and spectroscopic modes. The imaging camera was retired in late 2009, since then the telescope has observed entirely in spectroscopic mode. Images were taken using a photometric system of five filters (named u, g, r, i and z). These images are processed to produce lists of objects observed and various parameters, such as whether they seem pointlike or extended (as a galaxy might) and how the brightness on the CCDs relates to various kinds of astronomical magnitude.

For imaging observations, the SDSS telescope used the drift scanning technique, but with choreographed variation of right ascension, declination, tracking rate, and image rotation

which allows the telescope to track along great circles and continuously record small strips of the sky The telescope's imaging camera is made up of 30 CCD chips, each with a resolution of 2048×2048 pixels, totaling approximately 120 megapixels. Using these photometric data, stars, galaxies, and quasars are also selected for spectroscopy. Every night the telescope produces about 200 GB of data.

In Python, The API can be imported with:

```
from astroquery.sdss import SDSS
```

2.2 Interacting with SDSS through Astroquery

2.2.1 Querying objects and region

We can use this site to navigate objects within the SDSS footprint using their coordinates or name

`query_crossid()` can be used to query the service,using the cross-identification web interface,and returns a table object.We use cross-Id when we have IDs and positions (RA/Dec) and we need SDSS-III cross-matches for each of our objects

`query_region()` is used to query a region around given coordinates and returns a table object. It is equivalent to the object cross-ID from the web interface

Example:

```
pos =
    coord.SkyCoord('0h8m05.63s+14d50m23.3s',
    frame='icrs')
xid = SDSS.query_region(pos,
    spectro=True)
print(xid)
```

2.2.2 Downloading data

If we want to download spectra and/or images for our match, we have all the information we need in the elements of "xid" from the above example.

`get_images()` is used to download an image from SDSS. Querying SDSS for images will return the entire plate. For subsequent analyses of individual objects. We can download and plot the image in the following way:

```
im = SDSS.get_images(matches=xid,
    band='r')
hdulist = im[0]
data = hdulist[0].data
plt.imshow(data)
```

`get_spectra()` is used to download spectrum from SDSS.

```
sp = SDSS.get_spectra(matches=xid)
```

The variables "sp" and "im" are lists of HDUList objects, one entry for each corresponding object in xid.

Note that in SDSS, image downloads retrieve the entire plate, so further processing will be required to excise an image centered around the point of interest (i.e., the object(s) returned by `query_region`).

`get_spectral_template()` is used to download spectral templates from SDSS DR-2.

Location: http://www.sdss.org/dr7/algorithms/spectemplates/

There 32 spectral templates available from DR-2, from stellar spectra, to galaxies, to quasars. To see the available templates, do: do:

```
from astroquery.sdss import SDSS print
    SDSS.AVAILABLE_TEMPLATES
```

3. SIMBAD

3.1 Introduction to SIMBAD

The purpose of SIMBAD is to provide information on astronomical objects of interest which have been studied in scientific articles. This states:

> The SIMBAD database is managed by the Centre de Données astronomiques de Strasbourg (CDS). The SIMBAD data base presently (June 2020) contains information for:
>
> - About 5,800,000 stars;
> - About 5,500,000 non-stellar objects (galaxies, planetary nebulae, clusters, novae and supernovae

SIMBAD was constructed to facilitate cross-referencing between different star catalogs and we can do so directly by clicking here, or may access it through the package *Astropy*.

3.2 How to query SIMBAD

In Python, The API can be imported with:

```
>>> from astroquery.simbad import Simbad
```

Then, for instance, say one wants to extract information about the Messier object 'M2', then,

```
>>> result_table =
    Simbad.query_object("m2")
>>> print(result_table)
```

Multiple queries can also be done like this:

```
>>> from astroquery import simbad
>>> targets =
    [m31,  m51  ,  omc1  , notatarget ]
>>> queries = [simbad.QueryId(x) for x
    in targets]
>>> result =
   simbad.QueryMulti(queries).
      execute(mirror=harvard)
>>> print result.table
```

3.3 How to query different types of data

3.3.1 Wildcard entries

So for instance to query messier objects from 1 through 9:

```
>>> from astroquery.simbad import Simbad
>>> result_table =
    Simbad.query_object("m [1-9]",
    wildcard=True)
>>> print(result_table)
```

3.3.2 Querying regions

Queries that support a cone search with a specified radius - around an identifier or given coordinates are also supported. If an identifier is used then it will be resolved to coordinates using online name resolving services available in *astropy*.

```
>>> from astroquery.simbad import Simbad
>>> result_table =
    Simbad.query_region("m81")
>>> print(result_table)
```

When no radius is specified, the radius defaults to 20 arcmin. A radius may also be explicitly specified - it can be entered either as a string that is acceptable by `astropy.coordinates.Angle` or by using the *Quantity* object from `astropy.units`:

```
>>> from astroquery.simbad import Simbad
>>> import astropy.units as u
>>> result_table =
    Simbad.query_region("m81", radius=0.1
    * u.deg)
>>> # another way to specify the radius.
>>> result_table =
    Simbad.query_region("m81",
    radius='0d6m0s')
>>> print(result_table)
```

3.4 Changing default settings

For the our purposes, this will suffice. However, we may customise the default settings in order to limit the data that is provided to us.

3.4.1 Changing the row limit

To fetch all the rows in the result, the row limit must be set to 0. However for some queries, results are likely to be very large, in such cases it may be best to limit the rows to a smaller number.

```
>>> from astroquery.simbad import Simbad
>>> Simbad.ROW_LIMIT = 15 #now any query
    fetches at most 15 rows
```

3.4.2 Changing the timeout

The timeout is the time limit in seconds for establishing connection with the SIMBAD server and by default it is set to 100 seconds. You may want to modify this - again you can do this at run-time if you want to adjust it only for the current session.

```
>>> from astroquery.simbad import Simbad
>>> Simbad.TIMEOUT = 60 #sets the
    timeout to 60s
```

4. Vizier

4.1 Introduction to VizieR

VizieR is a Catalogue service, an API. The site itself was built by the University of Strasbourg. Official Documentation here states:

> VizieR provides the most complete library of published astronomical catalogues with verified and enriched data, accessible via multiple interfaces.

Query tools allow the user to select relevant data tables and to extract and format records matching given criteria. One can use the service either directly from here, or access them through the package *Astropy*, which was done in this project. In Python, The API can be imported with:

```
from astroquery.vizier import Vizier
```

4.2 Interacting with VizieR through Astroquery

4.2.1 Finding Catalogues

Currently, 21171 catalogues are available that can be accessed by VizieR.
To find them with keywords, `find_catalogs()` can be used.
Here, We need to find Henry Draper's catalogue.

```
catalogue_example =
    Vizier.find_catalogs('draper')
for K, V in catalogue_example.items():
   print(K, ':\t', V.description, '\n')
```

This prints out all catalogues with the keyword *draper* in their keys. One can easily pick out the catalogue in need from the description. The output of interest was:
III/135A : Henry Draper Catalogue and Extension (Cannon+ 1918-1924; ADC 1989)

4.2.2 Viewing Catalogues

Once you know the key/value of the catalogue of interest, `get_catalogs()` provides a convenient way to access that.

```
henryDraper =
    Vizier.get_catalogs('III/135A') # or
    value instead of the key
print(henryDraper)
henryDraper._dict
```

This prints out the number of tables in the catalogue (one in henryDraper) with rows, columns specified. _dict prints out the preview of the table in context.

4.2.3 Querying Objects

Any celestial object can be handled with VizieR, given that it's documented in the catalogues.

```
m13 = Vizier.query_object('m13')
print(m13)
```

This returns a list of relevant tables to the object. Once you find the relevant table's key, accessing is just like *OrderedDict* in Python.

4.2.4 Querying Regions

To query a region either the coordinates or the object name around which to query should be specified along with catalogue to refer to and the value for the radius (or height/width for a box) of the region, where the radius is provided in angles. If only one angle is provided, then region is treated as a square, else height and width are needed.
query_region method is used for this.

```
m13 = Vizier.query_region('m13',
    radius="0d6m0s", catalog="GSC")
# or
from astropy.coordinates import Angle
m13 = Vizier.query_region('m13',
    radius=Angle(0.1, "deg"),
    catalog="GSC")
# or
import astropy.units
m13 = Vizier.query_region('m13',
    radius=0.1*astropy.units.deg,
    catalog="GSC")
# or
m13 =
    Vizier.query_region(ra="16h41m42s",
    dec="36d27m41s", radius="0d6m0s",
    catalog="GSC")
```

4.2.5 Constraining Results

To do so, an instance of VizierClass class, with the conditions, must be initiated first. All further queries may then be performed on this instance rather than on the Vizier class.

```
vizClass = Vizier(columns=['B-V',
    'Vmag', 'Plx', '_DEJ2000'],
    columns_filters={"Vmag":">10"},
    keywords=["optical"])
```

Now we can call different query methods on this Vizier instance.

```
result = Vizier.query_object("HD
    226868", radius="20s",
    catalog=["NOMAD", "UCAC"])
print(result[0]) # no query
v = Vizier(columns=["*", "+_r"],
    catalog="II/246")
result = v.query_region("HD 226868",
    radius="20s")
print(result[0]) # Sorted data on basis
    of distance column "_r"
```

4.2.6 Querying Tables

A Table can be used to specify coordinates in a region query only if it contains *_RAJ2000* and *_DEJ2000* where J2000 is the currently used standard epoch, the Gregorian date January 1, 2000 at 12:00.
Example of querying a table is as follows, context is analysing *AGNs* in *Veron* and *Cety* catalogs with $10.0 < \text{VMAG} < 11.0$.

```
agn = Vizier(catalog="VII/258/vv10",
    columns=['*', '_RAJ2000','_DEJ2000'])
.query_constraints(Vmag="10.0..11.0")[0]
print(agn)
```

Case Study: The Pleiades Star Cluster

Abstract
After months of learning various astronomical and computational techniques we decided to put them on a test by doing a case study on a actual astronomical object. We will also learn about HR diagrams

[1] *Astronomy Club, IITK*
[2] *Science and Technology Council, IITK*

Contents

1. Introduction

The Primary objective of our project under Astronomy Cub IITK was to examine a catalog ,extract data from it and present our findings and results with appropriate plots and conclusions. We learned lot of essential skills to navigate through an astronomical catalog as well as methods to break down and conclude data comprising of thousands of sources. To test our skills we decided to do a case study on one of the many astronomical objects. for this we have considered the "Seven Sisters" or the Pleiades Cluster form the coveted Gaia Archives.

1.1 Pleiades Cluster

The Pleiades Cluster,also called the *Seven Sisters* or M45 is an open cluster of stars,nearest to the earth and most easily observable from the night sky. It belongs to the constellation Taurus and is composed of many B-class stars. The majority of the stars are blue and hot and have been predicted to have formed around 100 million years ago. There as many as 987 stars in the cluster.

1.2 Project Details

Our project involved extracting the Pleiades cluster data from the Gaia Archive, compiling and sorting it to present accurate results of the stellar Distance, Visual Magnitude, Luminosity and the Radius of the stars present in the cluster.
Further,we also attempted to predict the age of the cluster using the Hertzsprung-Russell Diagram.

Figure 1. M45: Pleiades Cluster

2. Process

2.1 Data Collection

Our first step was to collect data about Pleiades star cluster from official catalogues, for our purposes we decided to choose the Gaia Archive as our data source. To extract data from the archive we took the help of the Astroquery and Astropy libraries in python. With this we created a CSV file containing data like G-band mean magnitude, Right ascension, Declination, Parallax, BP - RP colour, stellar luminosity, stellar effective temperature and stellar radius.

2.2 Absolute Magnitude and Distance

The next task was to calculate the absolute magnitude (M) and the stellar distance (d) of the cluster. So, we used the following formula:

$$m - M = 5log(d) - 5 \quad (1)$$

	gmag	ra	dec	plx	bp_rp	lum_val	teff_val	radius_val
0	15.684173	58.452096	23.485778	3.486630	2.192428	0.062333	3797.0000	0.576938
1	7.498167	55.930096	25.080502	8.027063	0.188913	NaN	8337.3330	NaN
2	16.102013	57.170842	23.237979	8.509836	3.116683	NaN	3838.1667	NaN
3	12.923352	56.999005	24.731093	6.493298	1.591153	0.156267	4456.1650	0.663230
4	16.454210	55.865759	24.270604	2.675075	2.346289	0.050944	3830.0100	0.512623

Figure 2. First five entries in the created database

Where, d = 1000/parallax
m = G-band mean magnitude
With this we found the absolute magnitude of every star in the cluster and the distance of the cluster which came out to be 444.705021 light years, which when cross checked with the actual values comes out to be very accurate.

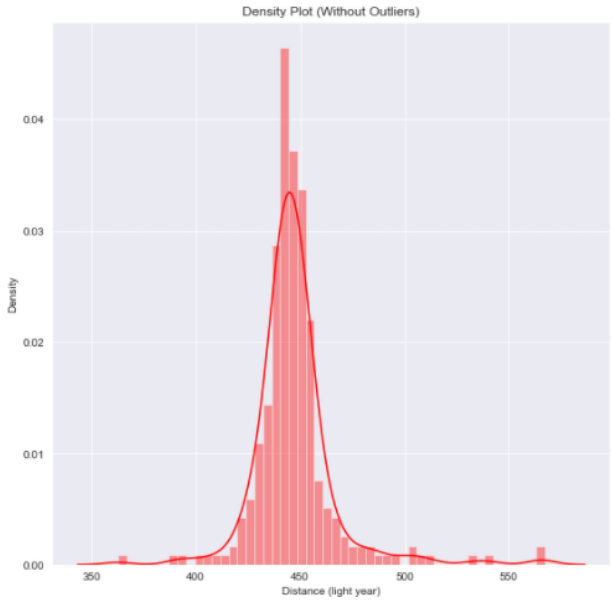


Figure 3. Density Plot with Distance

2.3 Luminosity and Temperature

Next we calculated the Luminosity and temperature of stars in the cluster and cross-checked them with their values in the database. Formulae used:

$$M = 4.77 - 2.5log(\frac{L}{L_0}) \tag{2}$$

$$T_k = \frac{5601}{(color + 9.4)^{\frac{2}{3}}} \tag{3}$$

We did a similar calculation for radius of the stars using the formula,

$$\frac{R}{R_0} = \frac{\sqrt{\frac{L}{L_0}}}{(\frac{T_k}{T_0})^2} \tag{4}$$

Next we plotted these values with their actual values from the database and observed the graph. On calculating the slope of

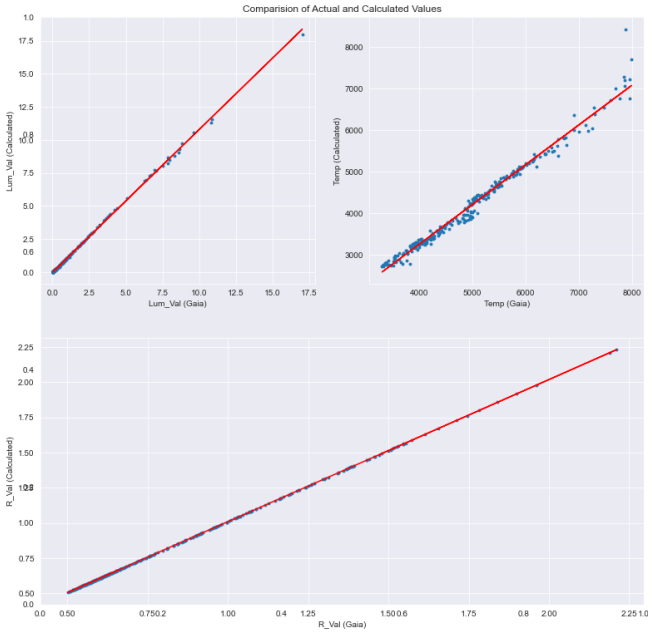


Figure 4. Comparison of calculated values with actual values

the above graphs we observed that each slope was near about equal to one which meant that our values were pretty accurate.

2.4 Position in the Night Sky

Then the next task is to plot the position of stars using right ascension and declination coordinates and we defined the size of the stars to be equal to their brightness to get a good idea of how the cluster actually looks. Brightness of a star is the difference of the maximum G-band magnitude and the G-band magnitude of that particular star.

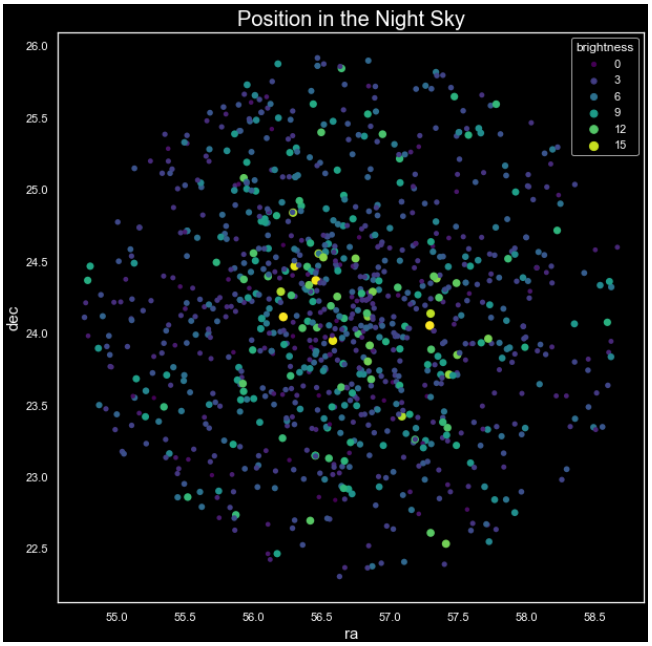


Figure 5. Position in the Night Sky

2.5 Hertzsprung-Russell Diagram

The Hertzsprung–Russell diagram, abbreviated as HR diagram , is a scatter plot of stars showing the relationship between the stars' absolute magnitudes or luminosities versus their stellar classifications or effective temperatures. The diagram was created independently around 1910 by Ejnar Hertzsprung and Henry Norris Russell, and represented a major step towards an understanding of stellar evolution.

Depending on its initial mass, every star goes through specific evolutionary stages dictated by its internal structure and how it produces energy. Each of these stages corresponds to a change in the temperature and luminosity of the star, which can be seen to move to different regions on the HR diagram as it evolves. This reveals the true power of the HR diagram – astronomers can know a star's internal structure and evolutionary stage simply by determining its position in the diagram.

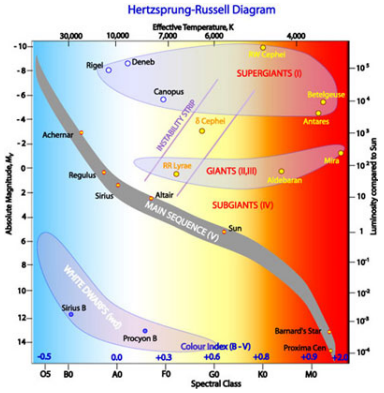


Figure 6. The Hertzsprung-Russell diagram at various stages of stellar evolution.

Most of the stars occupy the region in the diagram along the line called the main sequence. During the stage of their lives in which stars are found on the main sequence line, they are fusing hydrogen in their cores. The H-R diagram can be used by scientists to roughly measure how far away a star cluster or galaxy is from Earth. This can be done by comparing the apparent magnitudes of the stars in the cluster to the absolute magnitudes of stars with known distances (or of model stars).

Our task was to plot the HR diagram of the Pleiades cluster and using that find the approximate age of the cluster.

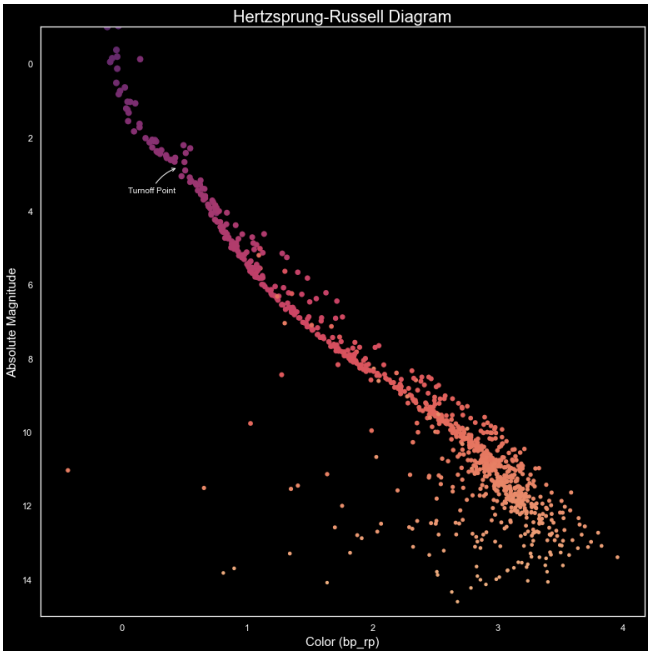


Figure 7. The Hertzsprung-Russell diagram of Pleiades Cluster

From the diagram we approximated the turnoff point (point where the HR diagram starts deviating from the main sequence). Now the formula for calculating the age of a cluster is:

$$t_{age} = t_0 \frac{L_0 M}{L M_0} \tag{5}$$

Where, $t_0 = 15 \times 10^9$ years
and, $L \propto M^3$
Putting in the values from the graph and database we get the age to be approximately 84 million years.

3. Conclusions

In this Finale project we applied all the concepts we have learnt so far in this span of 5 Weeks. We learnt different types of python libraries related to astrophysics like

- **Astropy**
- **Scipy**

- **Astroquery**

and and we used some basic libraries of python like :

- **Matplotlib**
- **Numpy**
- **Pandas**
- **Seaborn**

for analysing our data of **Pleiades Cluster** also known as Seven sisters. and plotting different kinds of plots like **density plot** and **Hertzsprung-Russell Diagram** We used the **Curve Fitting** techniques for plotting our data.
We Compared the values of luminosity,temperature and radius, that we derived in equations *2,3* and *4*, with the actual values of the stars using the data given.
We plotted a HR diagram and found the turnoff point using our knowledge. Then we estimated the age of the star cluster using equation *5*.

	gmag	ra	dec	plx	bp_rp	lum_val	teff_val	radius_val	abs_mag
0	15.684173	58.452096	23.485778	3.486630	2.192428	0.062333	3797.0000	0.576938	7.609925
1	7.498167	55.930096	25.080502	8.027063	0.188913	NaN	8337.3330	NaN	11.626516
2	16.102013	57.170842	23.237979	8.509836	3.116683	NaN	3838.1667	NaN	2.730650
3	12.923352	56.999005	24.731093	6.493298	1.591153	0.156267	4456.1650	0.663230	7.261571
4	16.454210	55.865759	24.270604	2.675075	2.346289	0.050944	3830.0100	0.512623	8.164680
...	...	...	...	...	...	...	...	...	...
982	11.852763	56.920869	22.929795	5.779793	1.404244	0.513834	4526.7550	1.165440	8.914174
983	8.186488	55.930264	24.374396	7.226433	0.352915	7.839014	7963.1953	1.470986	11.463561
984	15.502478	56.525937	25.112616	2.085063	1.515169	0.141687	4443.0000	0.635280	10.362302
985	17.422070	57.441505	22.895388	1.730391	2.049238	NaN	NaN	NaN	9.374970
986	11.906063	57.765432	22.840395	4.196960	1.102659	0.816104	4947.1800	1.229731	10.460912

Figure 8. Data of Pleiades Cluster

CONCLUSION

We wind up this project with lots of takeaways. Apart from getting to know the basic tools that an astrophysicist must have, we used like python, git, jupyter and anaconda to convert raw data to more understandable forms. We learnt about exciting celetial objects and mathematical techniques: Fourier transformation, binary starts and cepheid variables. We also processed images and extract essential information regarding a star's luminosity. For several of us, this means an initiation intro the scientific techniques used to study celestial objects, for others, it meant using a new programming language and thinking about how data behaves. Needless to say, this project enriched both our understanding of theory and well as where it is applied.

ACKNOWLEDGMENT

MENTORS

- Gurbaaz Singh Nandra
- Mohammad Saad
- Varun Muralidharan

MENTEES

- Nikita Singh
- Arkachur Bhattacharya
- Shreyansh Agarwal
- Ishita Agarwal
- Aryan Vora
- Sunreet
- B.Anshuman
- Rishi
- Sushmita
- Jhaansi Reddy
- Kalash Talati
- Ishita Vyavahare
- Divyam Jain
- Harish Prasad
- Divya M
- Shivang Pandey
- Kavish Priolkar
- Anjali Jain
- Sheshank
- Prabhdeeep Kaur