Exercise2_wbstyle

March 24, 2025

1 Exercise 2

1.0.1 Importing necessary libraries

In this step, we will import essential Python libraries for numerical computations, data visualization, and astronomy-related data handling:

```
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
from astropy.table import Table
from astropy import units as u
plt.ion()
import os
```

1.0.2 Working with Physical Quantities and Units

In this step, we will show how to defining physical quantities using Astropy units (`u`):

```
a = 50.0 * u.meter
b = (23, 45, 88) * u.meter
print(a, b)
```

1.0.3 Calculating the Mean of Quantities with Units

Now, we will compute the average (mean) value of the physical quantities stored in the array `b`:

```
np.mean(b)
```

1.0.4 Performing Unit-Aware Arithmetic Operations

In this step, we will calculate a velocity by dividing distance by time, explicitly handling units with Astropy:

```
15 * u.meter / (3 * u.second)
```

1.0.5 Defining and Displaying Astronomical Distances

Next we will define a distance using parsecs, a common astronomical unit of measurement:

```
x = 62 * u.parsec
print(x)
```

1.0.6 Defining Another Astronomical Distance

In this step, we will define a second astronomical distance variable (`y`) using parsecs as units:

```
y = 45 * u.parsec
```

1.0.7 Calculating the Ratio Between Two Astronomical Distances

We will calculate the dimensionless ratio of two distances (`x` and `y`), both defined in parsecs:

```
x / y
```

1.0.8 Extracting the Numerical Value from a Quantity with Units

Now, we will extract only the numerical component from the distance variable `x`, stripping away its associated units:

```
z = x.value
z
```

1.0.9 Calculating the Ratio Using Numerical Values (Without Units)

Let's calculate the numerical ratio of the values of two distances (`x` and `y`) after removing their associated units:

```
z = x.value / y.value
```

1.0.10 Rounding the Numerical Result

In this step, we will round the numerical value (`z`) to two decimal places:

```
np.around(z, decimals=2)
```

1.0.11 Checking the Data Type of a Variable

Next we will check and display the Python data type of the variable `x`:

```
type(x)
```

1.0.12 Calculating Speeds from Distance and Time Arrays

In this step, we will calculate speeds using arrays of distances (`b`) and corresponding times (`time`), handling units explicitly with Astropy:

```
time = [10, 20, 30] * u.second
print(time)
```

speed = b / time print(speed)

1.0.13 Plot of Wind Speed vs. Time

The plot below demonstrates the relationship between wind speed and time, calculated from previously defined data arrays. Each point represents a specific measurement of wind speed (in meters per second) over corresponding time intervals (in seconds). Additionally, a dashed gray line (`y =`

x) is included as a reference, helping visualize the distribution of the measurements relative to this reference line.

```
plt.figure(figsize=(5,3))
plt.plot(speed, time, ls='', color='#300500', marker='.', label='Wind Speed [m/s]')
plt.xlabel("Speed [m/s]", fontsize=14)
plt.ylabel("Time [s]", fontsize=14)
plt.legend()

l1 = np.linspace(0,100, 2)
l2 = np.linspace(0,100, 2)

plt.plot(l1, l2, color="gray", ls='--')
```

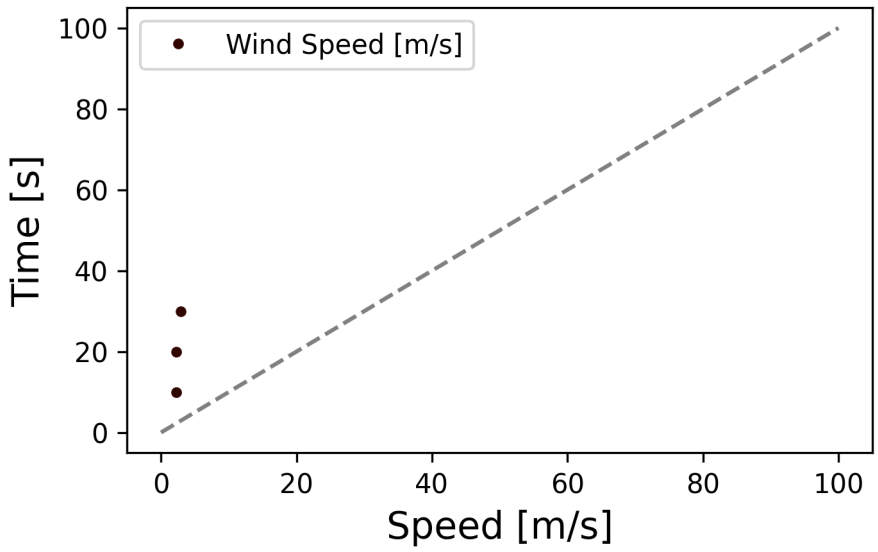

Figure: Scatter plot showing wind speed versus time. Each dot represents a measured wind speed (in m/s) at a given time (in seconds). The dashed gray line represents the `y = x` reference, highlighting how the data points deviate from a one-to-one relationship between speed and time.