Project One: Calculating the Rotation Curve of the Milky Way

Maggie Ramsey, Ainsley Jacquemain, and Lisa Chiang

The Motivation.

$$v = \sqrt{\frac{GM}{r}}$$

- Why aren't stars slowing down?
  - As radius decreases, we expect to see an appreciable decrease in the velocity of stars and bodies orbiting the Milky Way
  - Instead, we see a plateau... why is that?
  - Dark Matter!
- Knowing what we know about orbital velocity as well as what data we can find for the Milky Way, let's find that plateau for ourselves and make our own conclusion on Dark Matter.

1.
Calculating the Bulge Component

Packages and Formula

```
[1] import astropy.constants as const # import a Python package to call the value of certain constants
    import astropy.units as u # import a Python package to call certain units, e.g., the mass of the Sun
    import numpy as np #lets us do math

[2] def CalculatingOrbitalVelocity (M,R):
      orbitalvelocity = np.sqrt(const.G * M / R) # The formula we will use to calculate orbital velocity
      return orbitalvelocity # The result is orbital velocity
    print (const.G) # Show G

    Name   = Gravitational constant
    Value  = 6.6743e-11
    Uncertainty  = 1.5e-15
    Unit  = m3 / (kg s2)
    Reference = CODATA 2018
```

G

Defining Inputs

Google

```
[8] M_bulge = 1.5 * 10**10 * u.solMass  # The solar mass of the bulge is 1.5 * 10**10
    print (M_bulge.to(u.kg)) #Show the mass of the bulge as kg (change solMass to kg to use the formula)

    2.9826148060470763e+40 kg

[7] R_arr = (np.arange(1,30) * u.kpc) # Use radii outside of bulge
    print(R_arr.to(u.km)) # Show the radii as km to use in formula

    [3.08567758e+16 6.17135516e+16 9.25703274e+16 1.23427103e+17
     1.54283879e+17 1.85140655e+17 2.15997431e+17 2.46854207e+17
     2.77710982e+17 3.08567758e+17 3.39424534e+17 3.70281310e+17
     4.01138086e+17 4.31994861e+17 4.62851637e+17 4.93708413e+17
     5.24565189e+17 5.55421965e+17 5.86278740e+17 6.17135516e+17
     6.47992292e+17 6.78849068e+17 7.09705844e+17 7.40562620e+17
     7.71419395e+17 8.02276171e+17 8.33132947e+17 8.63989723e+17
     8.94846499e+17] km
```

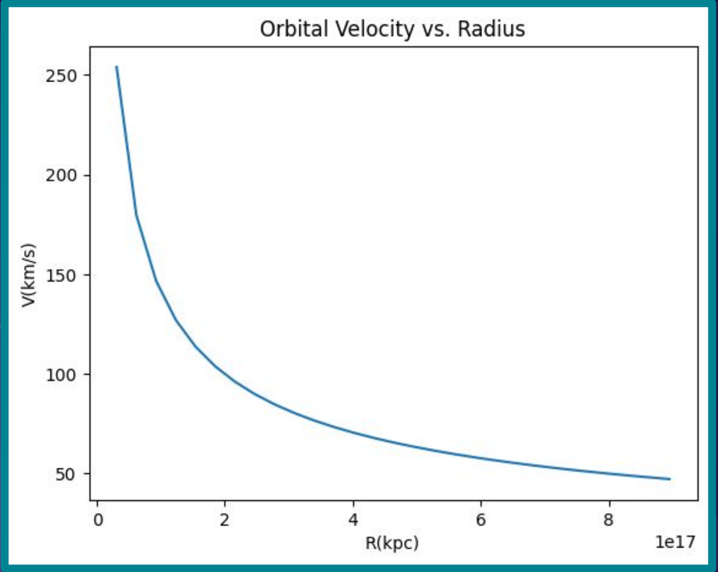
Calculating Velocities

```
[9] v_bulge = CalculatingOrbitalVelocity (M_bulge, R_arr) #calculate the orbital velocity using the formula
    print (v_bulge.to (u.km/u.s)) # Show the velocities with the unit km/s

    [253.99558864 179.60200312 146.64442148 126.99779432 113.59028044
     103.69326485  96.00130881  89.80100156  84.66519621  80.32045757
      76.58255145  73.32221074  70.44570143  67.88317646  65.58137899
      63.49889716  61.60297885  59.86733437  58.27058437  56.79514022
      55.42638148  54.15204145  52.96174134  51.84663242  50.79911773
      49.81263319  48.88147383  48.0006544   47.16579673] km / s
```

Plotting Data + Use of AI

```
[10] import matplotlib.pyplot as plt #import package to add plot
     plt.plot (R_arr.to(u.km), v_bulge.to(u.km/u.s)) #Define the axes as Radius and Velocity
     plt.xlabel ("R(kpc)") # Name the x axis
     plt.ylabel ("V(km/s)") # Name the y axis
     plt.title ("Orbital Velocity vs. Radius") # Name the plot
     plt.show () #show the plot
```



★ Gemini AI was used here to generate this graph!

2.
Calculating the Disk Component

Defining Inputs

```
# Define the disk's total mass (M_disk)
M_disk = 1.25**2 * 1e11 * u.solMass
# Define the outer edge of the disk's radius
R_disk = 10 * u.kpc
#Calculating the density in solar masses per square kpc
density_disk = M_disk/(np.pi*(R_disk**2))
print(density_disk.to(1e6 * u.solMass/u.kpc**2)) #printing the density calculated for future use
```

Results: 497.3591971621729 1e+06 solMass / kpc2

Defining Function: Calculating Enclosed Mass For the Disk

```
def calculatingEnclosedMassForDisk(R, density=497 * 1e6 * u.solMass/u.kpc**2):
    """
    Calculate enclosed mass for the disk component
    Input: R - orbital radius, density - density of the disk as calculated above
    Output: M - enclosed mass
    """
    if R < 10 * u.kpc:
        M = np.pi * (R**2) * density
    else:
        # any radius larger than 10 kpc will be trucated at 10 kpc because of the extent of the disk component
        R = 10 * u.kpc
        M = np.pi * (R**2) * density
    return(M)
```

Visualize the Results

```
print(calculatingEnclosedMassForDisk(1 * u.kpc).to(1e6 * u.solMass), "at 1 kpc")
print(calculatingEnclosedMassForDisk(5 * u.kpc).to(1e6 * u.solMass), "at 5 kpc")
```

Results:1561.3715488341275 1e+06 solMass at 1 kpc
39034.28872085318 1e+06 solMass at 5 kpc

Converted to million solar masses for readability

Defining Function: Calculating the Enclosed Mass For the Milky Way

```python
def calculatingEnclosedMassForMilkyWay(R, density_disk=497 * 1e6 * u.solMass/u.kpc**2, M_bulge = 1e10 * u.solMass):
    """
    Note that the halo mass is missing here, so that is what you will work on this Friday
    """
    M_disk = calculatingEnclosedMassForDisk(R, density=density_disk)
    M_bulge = M_bulge
    M_total = M_disk + M_bulge
    return(M_total)
```

```python
# redefine the bulge orbital velocity array to v_bulge_arr to avoid confusion
v_bulge_arr = CalculatingOrbitalVelocity(M_bulge, R_arr)
```

Final Calculations

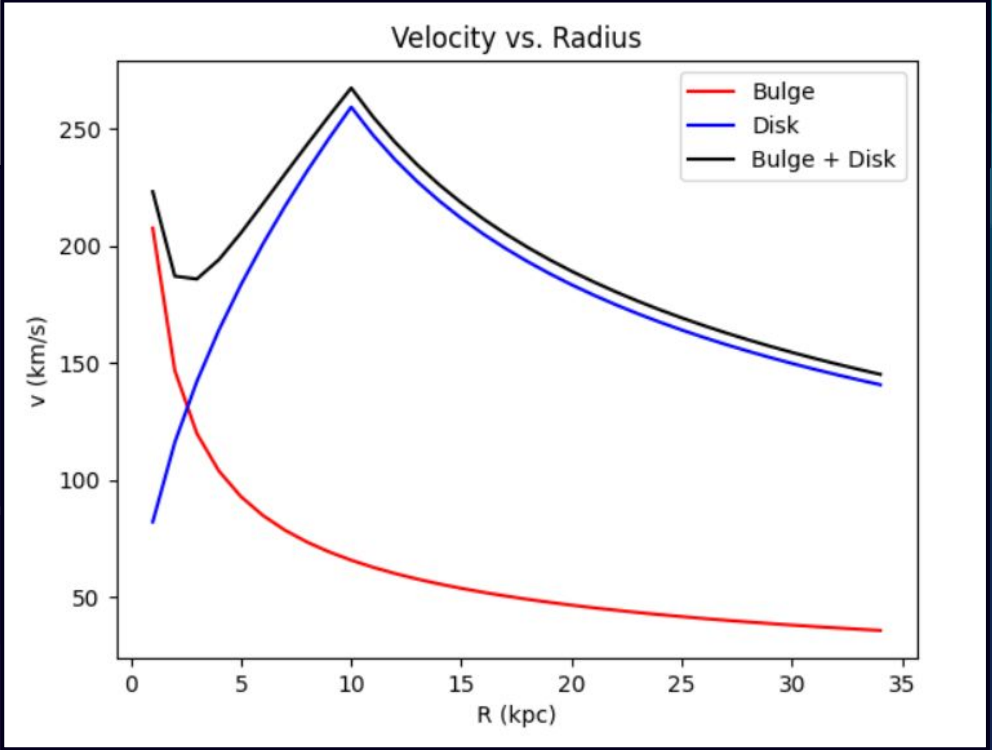
```
# Define arrays to be calculated
M_disk = np.zeros(len(R_arr)) * u.solMass # Don't forget unit here
M_total = np.zeros(len(R_arr)) * u.solMass

# Calculating enclosed masses for the disk and total component
for i in np.arange(len(R_arr)):
    M_disk[i] = calculatingEnclosedMassForDisk(R_arr[i])
    M_total[i] = calculatingEnclosedMassForMilkyWay(R_arr[i])

# Calculating the resulting orbital velocity due to each component
v_disk_arr = CalculatingOrbitalVelocity(M_disk, R_arr)
v_bulge_disk_arr = CalculatingOrbitalVelocity(M_total, R_arr)
```

Plotting Data

```
# Plot them all
plt.plot(R_arr.to(u.kpc), v_arr.to(u.km/u.s), color="red", label="Bulge") # note that here we can add label to the rotation curve
plt.plot(R_arr.to(u.kpc), v_disk_arr.to(u.km/u.s), color="blue", label="Disk") # plotting disk
plt.plot(R_arr.to(u.kpc), v_bulge_disk_arr.to(u.km/u.s), color="black", label="Bulge + Disk") # plotting bulge + disk components
plt.xlabel('R (kpc)') #label of the x axis
plt.ylabel('v (km/s)') #label of the y axis
plt.title("Velocity vs. Radius") #tiltle of the plot
plt.legend() # to show the legend of a figure
plt.show()
```



3.
Calculating the Halo Component

Defining Inputs

```
[15] M_halo = 1e12 * u.solMass # define halo total mass.
     R_halo = 50 * u.kpc # define the outer edge of halo radius.
     density_halo = M_halo / (np.pi * (4 / 3) * (R_halo**3)) # define the density is in the unit of solar mass per square kpc.
     print(density_halo.to(1e6 * u.solMass/u.kpc**3)) # check calculations by printing along the way.

1.9098593171027443 1e+06 solMass / kpc3
```

- Define known inputs
  - Remember that $V = \sqrt{(GM/R)}$
  - Remember that G was previously defined in the code.
  - Spherical density– we will do more with this later on :)
- Check as you go by printing values!

Halo Mass Function

```
def calculatingEnclosedMassForHalo (R, density_halo = 1.91 * 1e6 * u.solMass/u.kpc**3): # R and density_halo are the inputs of the function.
    """
    Input:
    R, the radius.
    Density_halo, the density of the halo used on each component of the array.
    Output:
    M, the enclosed mass.
    """
    M = np.pi * (4 / 3) * (R**3) * density_halo
    return(M)
```

+ Code + Text

```
[17] print(calculatingEnclosedMassForHalo(1 * u.kpc, density_halo).to(1e6 * u.solMass), "at 1 kpc") # convert to 1e6 solar masses to increase readability.
     print(calculatingEnclosedMassForHalo(5 * u.kpc, density_halo).to(1e6 * u.solMass), "at 5 kpc")
```

```
8.000000000000002 1e+06 solMass at 1 kpc
1000.0000000000003 1e+06 solMass at 5 kpc
```

- Make a function for enclosed mass
  - Parts: Input, output, definitions as needed, 'return'
  - No truncation / 'if statement' this time :(
- Check as you go by printing!

New Milky Way Mass Function

```python
[18] def calculatingNewEnclosedMassForMilkyWay(R, density_disk = 497.36 * 1e6 * u.solMass/u.kpc**2, M_bulge = 1e10 * u.solMass): # R, density_disk, and M_bulge are
     # While odd, density_halo is not in the above values because the differences in units between density_disk and density_halo cause issues in the code.
         """
         Input:
         R, the radius.
         Density_disk, the density of the disk used on each component of the array.
         Density_halo, the density of the halo used on each component of the array.
         M_bulge, the mass of the bulge.
         M_disk, the mass of the disk (which is not given as a value in the line above, because it must be calculated for radii each component of the array).
         M_halo, the mass of the halo (which is not given as a value in the line above, because it must be calculated for radii each component of the array).
         Output:
         M, the total enclosed mass for the Milky Way up to this point.

         When finding orbital velocity, sum the masses, then calculate velocity.
         """
         M_disk = calculatingEnclosedMassForDisk(R, density_disk)
         M_halo = calculatingEnclosedMassForHalo(R, density_halo)
         M_total = M_disk + M_bulge + M_halo # M_total is a global variable, not a local variable, so we can reuse it. Yay!
         return(M_total)
     calculatingNewEnclosedMassForMilkyWay(R_halo) # continue to check along the way.
```

$1.1662503 \times 10^{12}\ M_\odot$

- Make a function for total enclosed mass
  - Parts: Input, output, definitions as needed, 'return'

Defining Arrays

```
[19] # Define the arrays to be calculated.
     M_halo = np.zeros(len(R_arr)) * u.solMass # specify unit.
     M_total2 = np.zeros(len(R_arr)) * u.solMass # specify unit.
     """
     This specification helps the code pass an array through calculatingEnclosedMassForDisk, possibly preventing bugs.
     np.zeros is meant to be a placeholder. It will generate an array of zeros. See printed arrays below.
     len(R_arr) says that the array of zeros will have the same number of values as R_arr, which is why we see 30 zeros below.
     """
     M_halo = np.zeros(len(R_arr)) * u.solMass # specify unit.
     print(M_halo)
     print(M_total2)
```

```
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] solMass
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.] solMass
```

- Define arrays for mass component
  - Help your code to function properly!
  - Use 'len' to transcribe radius array
- Check as you go by printing– visual representations help!

Calculating Velocities

```
[20] # Calculate the enclosed masses for the disk radial components and the total radial components.
     for i in np.arange(len(R_arr)):
         M_halo[i] = calculatingEnclosedMassForHalo(R_arr[i], density_halo)
         M_total2[i] = calculatingNewEnclosedMassForMilkyWay(R_arr[i])
         print(R_arr[i], M_halo[i])
     """
     'for i in:' is a control flow statement called a 'for loop'.
     i goes through each array value and is calculatingEnclosedMassforHalo, making an array for M_disk, and an array for M_total2.
     """
     # Calculate the resulting orbital velocity due to each component.
     v_halo_arr = CalculatingOrbitalVelocity(M_halo, R_arr)
     v_bulge_disk_halo_arr = CalculatingOrbitalVelocity(M_total2, R_arr)
```

- Apply those brand new arrays!
  - Use a 'for loop' to apply the function to the new arrays
  - Now is the time when it is crucial to have different names for all of your various local variables...
- Check as you go by printing!

Adding Error Bars

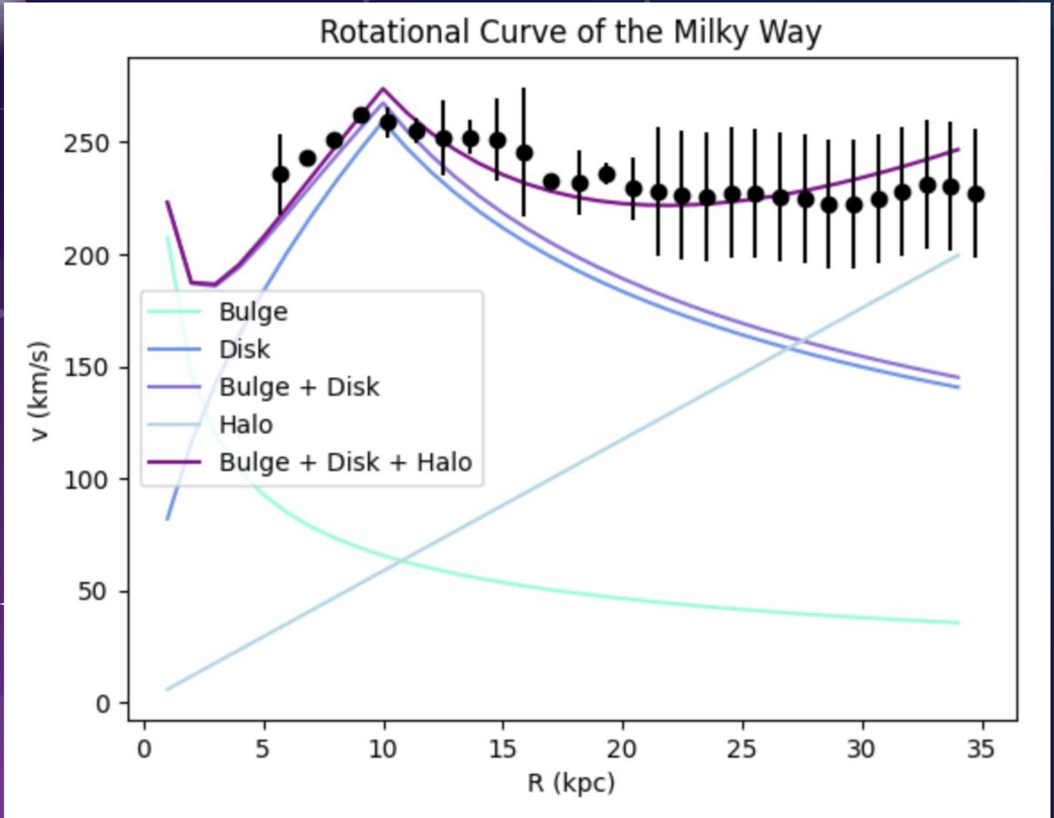
```
[21] from google.colab import files
     uploaded = files.upload()
     # Use this to upload a file into the code. In our case, the data provided in the file makes an overplot for the plot below.
```

```
Choose Files  galaxy_rota...n_2006.txt
• galaxy_rotation_2006.txt(text/plain) - 643 bytes, last modified: n/a - 100% done
Saving galaxy_rotation_2006.txt to galaxy_rotation_2006.txt
```

```
[22] import astropy.io.ascii
     tab = astropy.io.ascii.read("galaxy_rotation_2006.txt")
     # In this sequence, a package reads the provided data in the file and saves it under 'tab', thus making it available for use in the plot below.
```

- Get ready to put error bars on your final plot!
  - Import a document of experimental data
  - Use astropy to read and store that data
- (You are going to accidentally re-run this code multiple times, and then to your dismay, have to re-import the data over and over. Oops!)

Final Graph



- Note the legend, labeled axes, and title.
- Consider the difference between "Bulge + Disk" and "Bulge + Disk + Halo"… what does this mean?

Thank You!