Gala: Galactic Dynamics

Gillian La Vina

1.

The package *Gala* aims to help study large-scale astronomical systems by modelling orbits and trajectories using numerical integration.

Gala is mainly oriented towards galactic dynamics research, but can also be used to study stars, dark matter particles, and other astronomical objects.

Gala uses C under the hood for fast computations. Gala is also affiliated with and heavily uses Astropy.

Some notable operations of Gala include finding gravitational potential and force, integrated orbits, transforming coordinates, and nonlinear dynamics computations.

This report uses Gala 1.9.1

In [1]:
```
import gala
print(gala.__version__)
```

```
1.9.1
```

2.

I selected this package because I am interested in astronomical research and studying large-scale structures in the future. Gala seemed directly relevant and straightforward. I am also interested in using programming for modelling interactions as well; currently for research I am learning to model particle interactions, and I hope to extend my modelling/simulations knowledge to large-scale structures and the gravitational interactions.

3.

Version 1.0 released on April 13, 2019. The first ever iteration, version 0.1.3, was released Feb 23 2017, and was published in The Journal of Open Source Software October 2017.

Gala does not appear to have direct geneology, but it is closely affiliated with Astropy.

The version I installed is 1.9.1 (see #1).

4

Gala is still maintained by its original creator.

There are instructions on how to contribute on the website; anyone can contribute via pull requests on GitHub. It is also open to feature requests and bug reports.

5

Installing Gala was straightforward using !pip install. It is also a requirement to install Astropy, as Gala uses it heavily. Most of the samples/tutorials on Gala's official website also uses other packages that would need to be installed (pyia, astroquery).

Gala is not hard to use from a programming standpoint, but requires a fair amount of astronomy/physics knowledge, such as reference frames and potentials.

6

Gala installed on my computer using the standard pip command.

7

The source code is available on Gala's GitHub page.

8

Gala does not appear to be a core dependency for any other well known Python package.

9

The code can be used in a Jupyter notebook, or any other place Python can be used.

11 - Figures

The package does not create figures on its own, but is commonly used with matplotlib.

10, 12 - Provide Examples, Create a Figure in Gala

Tutorial: Galactic orbit for a star using Gaia data - Integrating the Sun's Orbit

This code is mostly taken from the example at https://gala.adrian.pw/en/latest/tutorials/pyia-gala-orbit.html, with a few extra graphs

In [2]:
```
# Some other required installs/downloads for this example

# Pyia data
!pip install pyia
# Astroquery
!pip install astroquery
```

```
Requirement already satisfied: pyia in /Users/maika/Library/jupyterlab-deskt
op/envs/env_1/lib/python3.12/site-packages (1.4.1)
Requirement already satisfied: astropy in /Users/maika/Library/jupyterlab-de
sktop/envs/env_1/lib/python3.12/site-packages (from pyia) (7.0.1)
Requirement already satisfied: numpy in /Users/maika/Library/jupyterlab-desk
top/envs/env_1/lib/python3.12/site-packages (from pyia) (2.1.0)
Requirement already satisfied: pandas in /Users/maika/Library/jupyterlab-des
ktop/envs/env_1/lib/python3.12/site-packages (from pyia) (2.2.2)
Requirement already satisfied: pyerfa>=2.0.1.1 in /Users/maika/Library/jupyt
erlab-desktop/envs/env_1/lib/python3.12/site-packages (from astropy->pyia)
(2.0.1.5)
Requirement already satisfied: astropy-iers-data>=0.2025.1.31.12.41.4 in /Us
ers/maika/Library/jupyterlab-desktop/envs/env_1/lib/python3.12/site-packages
(from astropy->pyia) (0.2025.4.7.0.35.30)
Requirement already satisfied: PyYAML>=6.0.0 in /Users/maika/Library/jupyter
lab-desktop/envs/env_1/lib/python3.12/site-packages (from astropy->pyia) (6.
0.2)
Requirement already satisfied: packaging>=22.0.0 in /Users/maika/Library/jup
yterlab-desktop/envs/env_1/lib/python3.12/site-packages (from astropy->pyia)
(24.1)
Requirement already satisfied: python-dateutil>=2.8.2 in /Users/maika/Librar
y/jupyterlab-desktop/envs/env_1/lib/python3.12/site-packages (from pandas->p
yia) (2.9.0)
Requirement already satisfied: pytz>=2020.1 in /Users/maika/Library/jupyterl
ab-desktop/envs/env_1/lib/python3.12/site-packages (from pandas->pyia) (202
4.1)
Requirement already satisfied: tzdata>=2022.7 in /Users/maika/Library/jupyte
rlab-desktop/envs/env_1/lib/python3.12/site-packages (from pandas->pyia) (20
24.1)
Requirement already satisfied: six>=1.5 in /Users/maika/Library/jupyterlab-d
esktop/envs/env_1/lib/python3.12/site-packages (from python-dateutil>=2.8.2-
>pandas->pyia) (1.16.0)
Requirement already satisfied: astroquery in /Users/maika/Library/jupyterlab
-desktop/envs/env_1/lib/python3.12/site-packages (0.4.10)
Requirement already satisfied: numpy>=1.20 in /Users/maika/Library/jupyterla
b-desktop/envs/env_1/lib/python3.12/site-packages (from astroquery) (2.1.0)
Requirement already satisfied: astropy>=5.0 in /Users/maika/Library/jupyterl
ab-desktop/envs/env_1/lib/python3.12/site-packages (from astroquery) (7.0.1)
Requirement already satisfied: requests>=2.19 in /Users/maika/Library/jupyte
rlab-desktop/envs/env_1/lib/python3.12/site-packages (from astroquery) (2.3
2.3)
Requirement already satisfied: beautifulsoup4>=4.8 in /Users/maika/Library/j
upyterlab-desktop/envs/env_1/lib/python3.12/site-packages (from astroquery)
(4.12.3)
Requirement already satisfied: html5lib>=0.999 in /Users/maika/Library/jupyt
erlab-desktop/envs/env_1/lib/python3.12/site-packages (from astroquery) (1.
1)
Requirement already satisfied: keyring>=15.0 in /Users/maika/Library/jupyter
lab-desktop/envs/env_1/lib/python3.12/site-packages (from astroquery) (25.6.
0)
Requirement already satisfied: pyvo>=1.5 in /Users/maika/Library/jupyterlab-
desktop/envs/env_1/lib/python3.12/site-packages (from astroquery) (1.6.2)
Requirement already satisfied: pyerfa>=2.0.1.1 in /Users/maika/Library/jupyt
erlab-desktop/envs/env_1/lib/python3.12/site-packages (from astropy>=5.0->as
troquery) (2.0.1.5)
Requirement already satisfied: astropy-iers-data>=0.2025.1.31.12.41.4 in /Us
```

```
ers/maika/Library/jupyterlab-desktop/envs/env_1/lib/python3.12/site-packages
(from astropy>=5.0->astroquery) (0.2025.4.7.0.35.30)
Requirement already satisfied: PyYAML>=6.0.0 in /Users/maika/Library/jupyter
lab-desktop/envs/env_1/lib/python3.12/site-packages (from astropy>=5.0->astr
oquery) (6.0.2)
Requirement already satisfied: packaging>=22.0.0 in /Users/maika/Library/jup
yterlab-desktop/envs/env_1/lib/python3.12/site-packages (from astropy>=5.0->
astroquery) (24.1)
Requirement already satisfied: soupsieve>1.2 in /Users/maika/Library/jupyter
lab-desktop/envs/env_1/lib/python3.12/site-packages (from beautifulsoup4>=4.
8->astroquery) (2.5)
Requirement already satisfied: six>=1.9 in /Users/maika/Library/jupyterlab-d
esktop/envs/env_1/lib/python3.12/site-packages (from html5lib>=0.999->astroq
uery) (1.16.0)
Requirement already satisfied: webencodings in /Users/maika/Library/jupyterl
ab-desktop/envs/env_1/lib/python3.12/site-packages (from html5lib>=0.999->as
troquery) (0.5.1)
Requirement already satisfied: jaraco.classes in /Users/maika/Library/jupyte
rlab-desktop/envs/env_1/lib/python3.12/site-packages (from keyring>=15.0->as
troquery) (3.4.0)
Requirement already satisfied: jaraco.functools in /Users/maika/Library/jupy
terlab-desktop/envs/env_1/lib/python3.12/site-packages (from keyring>=15.0->
astroquery) (4.1.0)
Requirement already satisfied: jaraco.context in /Users/maika/Library/jupyte
rlab-desktop/envs/env_1/lib/python3.12/site-packages (from keyring>=15.0->as
troquery) (6.0.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /Users/maika/Libr
ary/jupyterlab-desktop/envs/env_1/lib/python3.12/site-packages (from request
s>=2.19->astroquery) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /Users/maika/Library/jupyterl
ab-desktop/envs/env_1/lib/python3.12/site-packages (from requests>=2.19->ast
roquery) (3.8)
Requirement already satisfied: urllib3<3,>=1.21.1 in /Users/maika/Library/ju
pyterlab-desktop/envs/env_1/lib/python3.12/site-packages (from requests>=2.1
9->astroquery) (2.2.2)
Requirement already satisfied: certifi>=2017.4.17 in /Users/maika/Library/ju
pyterlab-desktop/envs/env_1/lib/python3.12/site-packages (from requests>=2.1
9->astroquery) (2024.7.4)
Requirement already satisfied: more-itertools in /Users/maika/Library/jupyte
rlab-desktop/envs/env_1/lib/python3.12/site-packages (from jaraco.classes->k
eyring>=15.0->astroquery) (10.7.0)
```

In [3]:
```
# Basic imports
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm

# Astropy imports
import astropy
import astropy.coordinates as coord
import astropy.units as u

# Gala
import gala.dynamics as gd
import gala.potential as gp
import gala.coordinates as gc
```

```
# Import GaiaData
from pyia import GaiaData
```

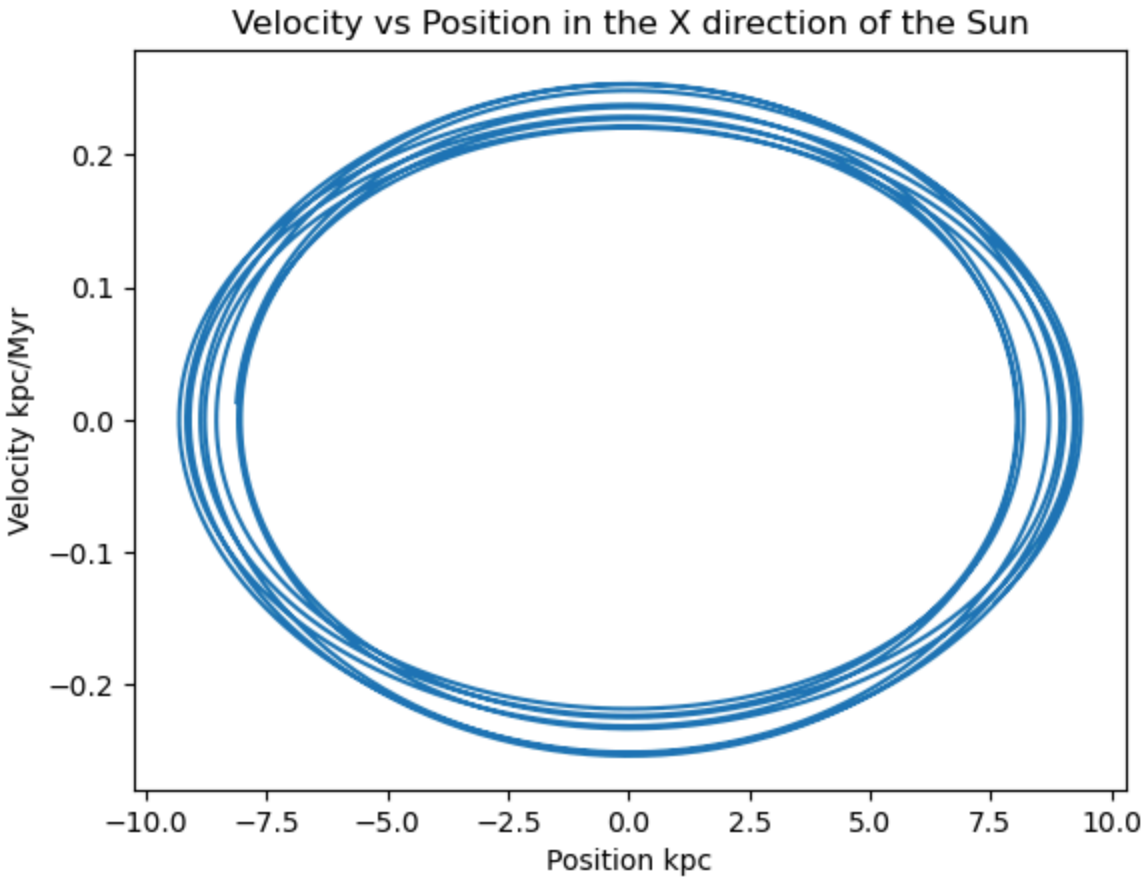
In [4]:
```
# Definte Galactocenteric Coordinate Frame

with coord.galactocentric_frame_defaults.set("v4.0"):
    galcen_frame = coord.Galactocentric()

# Some info for the frame:
# Describes galactic center from astropy
#galcen_frame?
```

In [5]:
```
# Getting the Orbit of the Sun

# Initial Conditions of Sun: Position and Velocity
    # Sun is along x-axis, galactic rotation is in y-direction
# Sun's Position
sun_pos = u.Quantity( # in x, y, z
                    [-galcen_frame.galcen_distance, # x
                    0 * u.kpc,                      # y
                    galcen_frame.z_sun]             # z
                    )
# Use solar velocity vector (astropy) to define sun's phase-space position
sun_w0 = gd.PhaseSpacePosition(pos=sun_pos,                  # position
                               vel=galcen_frame.galcen_v_sun  # velocity
                              )


# Now specify a mass/potential model for Galaxy: use Milky Way Potential
milk_pot = gp.MilkyWayPotential() # Potential of the milky way; 4-component


# Set time scale
dt = 0.5 * u.Myr
t1 = 0 * u.Myr
t2 = 2 * u.Gyr

# Integrate the Orbit
sun_orb = milk_pot.integrate_orbit(sun_w0,
                                   dt = dt,   # dt is every 0.5 Myr
                                   t1 = t1,            # start at 0 Gyr
                                   t2 = t2,      # end at 4 Gyr
                                  )
# Investigate sun_orb object
print(type(sun_orb))
print(sun_orb.pos)
print(sun_orb.pos.x)
print(sun_orb.vel)

# Plotting velocity over position
fig, ax = plt.subplots()
x = sun_orb.pos.x
vx = sun_orb.vel.d_x
ax.plot(x, vx)
```

```
ax.set_title("Velocity vs Position in the X direction of the Sun")
ax.set_xlabel("Position kpc")
ax.set_ylabel("Velocity kpc/Myr")
```

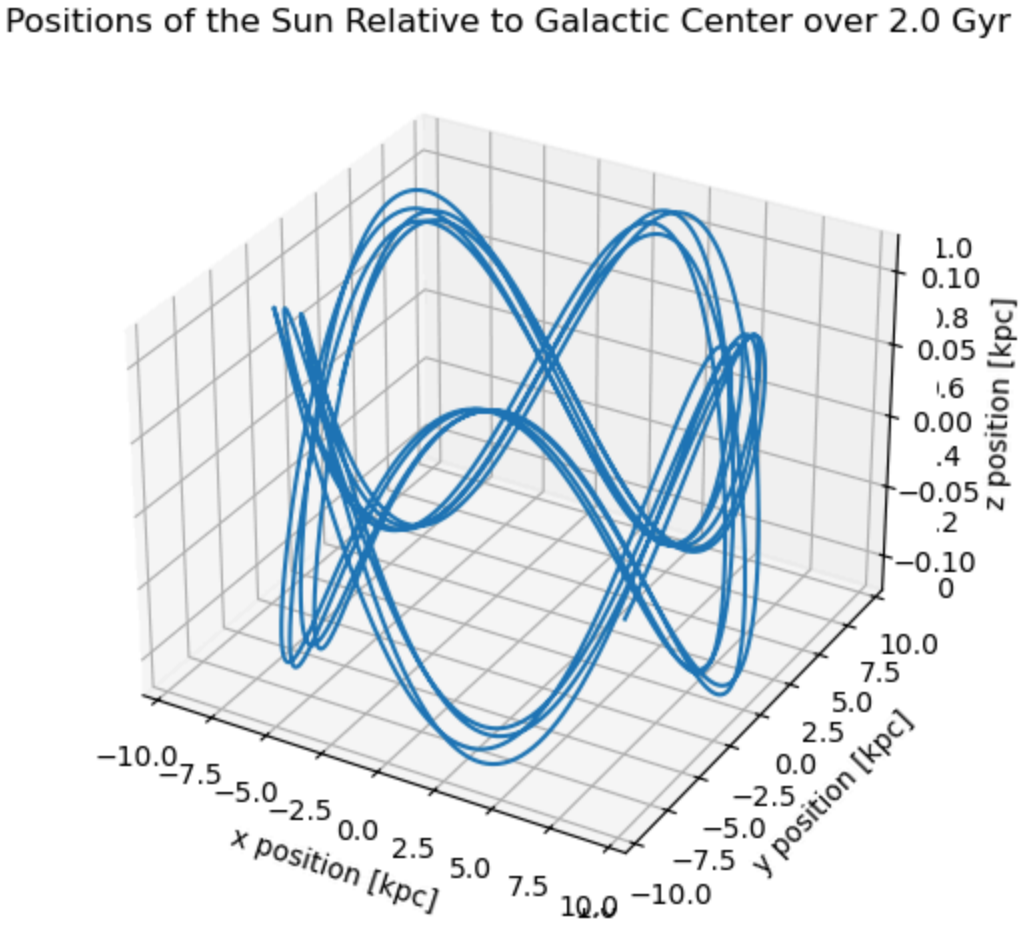
```
<class 'gala.dynamics.orbit.Orbit'>
[(-8.122     ,  0.        ,  0.0208    ),
 (-8.11454074,  0.12558905,  0.0247635 ),
 (-8.10535462,  0.25115138,  0.02869159), ...,
 ( 7.73046066, -5.07911294, -0.02079661),
 ( 7.66345746, -5.16704011, -0.02443138),
 ( 7.59522167, -5.25413621, -0.02804109)] kpc
[-8.122      -8.11454074 -8.10535462 ...  7.73046066  7.66345746
  7.59522167] kpc
[( 0.01319299,  0.25117811,  0.0079567 ),
 ( 0.01664538,  0.25115138,  0.00789159),
 ( 0.02009995,  0.25107112,  0.00781514), ...,
 (-0.13276506, -0.17666992, -0.00729087),
 (-0.13523898, -0.17502327, -0.00724449),
 (-0.13769523, -0.17334571, -0.00719062)] kpc / Myr
```

Out[5]: Text(0, 0.5, 'Velocity kpc/Myr')



In [6]:
```
# A position plot in 3D!

fig2, ax2 = plt.subplots(subplot_kw={"projection": "3d"})
ax2 = plt.axes(projection='3d')
ax2.plot3D(sun_orb.pos.x, sun_orb.pos.y, sun_orb.pos.z);

ax2.set_title(f"Positions of the Sun Relative to Galactic Center over {t2}")
ax2.set_xlabel("x position [kpc]")
```

```
ax2.set_ylabel("y position [kpc]")
ax2.set_zlabel("z position [kpc]")

fig2.tight_layout();
```



In [7]:
```
# Retrieve Gaia Data for Kepler-444 (comparison star)

# Retrieve sky position of Kepler-444 using SkyCoord.from_name() which queri

star_sky_c = coord.SkyCoord.from_name("Kepler-444")
print(star_sky_c)
print(type(star_sky_c))

#star_sky_c?
```

```
<SkyCoord (ICRS): (ra, dec) in deg
    (289.75228708, 41.63460623)>
<class 'astropy.coordinates.sky_coordinate.SkyCoord'>
```

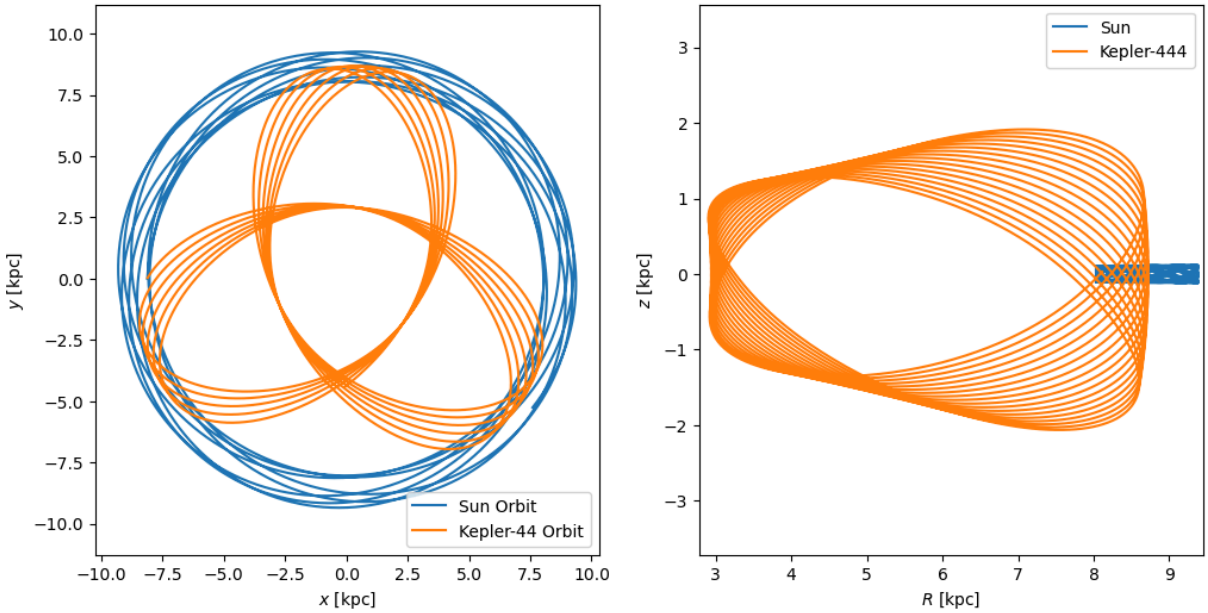
In [8]:
```
# Positional cross-match query on Gaia catalog

star_gaia = GaiaData.from_query(
    f"""
    SELECT TOP 1 * FROM gaiadr3.gaia_source
    WHERE 1=CONTAINS(
        POINT('ICRS', {star_sky_c.ra.degree}, {star_sky_c.dec.degree}),
        CIRCLE('ICRS', ra, dec, {(15*u.arcsec).to_value(u.degree)})
    )
```

```
    ORDER BY phot_g_mean_mag
    """
)

#star_gaia?
```

In [9]:
```
# Convert star_gaia measurements into an Astropy SkyCoord object (with posit

star_gaia_c = star_gaia.get_skycoord()

# Then, to compute this comparison star's galactic orbit, we convert the dat
# so Heliocentric) into the Galactocentric coordinate frame;

# Use astropy.coordinates transformation capabilities:

star_galcen = star_gaia_c.transform_to(galcen_frame)
#star_galcen?
```

In [10]:
```
# Initial conditions for the star
star_w0 = gd.PhaseSpacePosition(star_galcen.data);
star_orbit = milk_pot.integrate_orbit(star_w0,
                                      t=sun_orb.t # same as Sun Orbit
                                     )
print(type(sun_orb))
print(type(star_orbit))

fig3, ax3 = plt.subplots(1, 2, figsize = (12, 6));

# Can plot directly from "orbit" class
sun_orb.plot(["x", "y"], axes=ax3[0], label = "Sun Orbit");
star_orbit.plot(["x", "y"], axes=ax3[0], label = "Kepler-44 Orbit");
ax3[0].legend();

# Cylindrical plots
sun_orb.cylindrical.plot(
    ["rho", "z"],
    axes=ax3[1],
    labels=["$R$ [kpc]", "$z$ [kpc]"],
    label="Sun",
)
star_orbit.cylindrical.plot(
    ["rho", "z"],
    axes=ax3[1],
    labels=["$R$ [kpc]", "$z$ [kpc]"],
    label="Kepler-444",
)

ax3[1].legend();
```

```
<class 'gala.dynamics.orbit.Orbit'>
<class 'gala.dynamics.orbit.Orbit'>
```

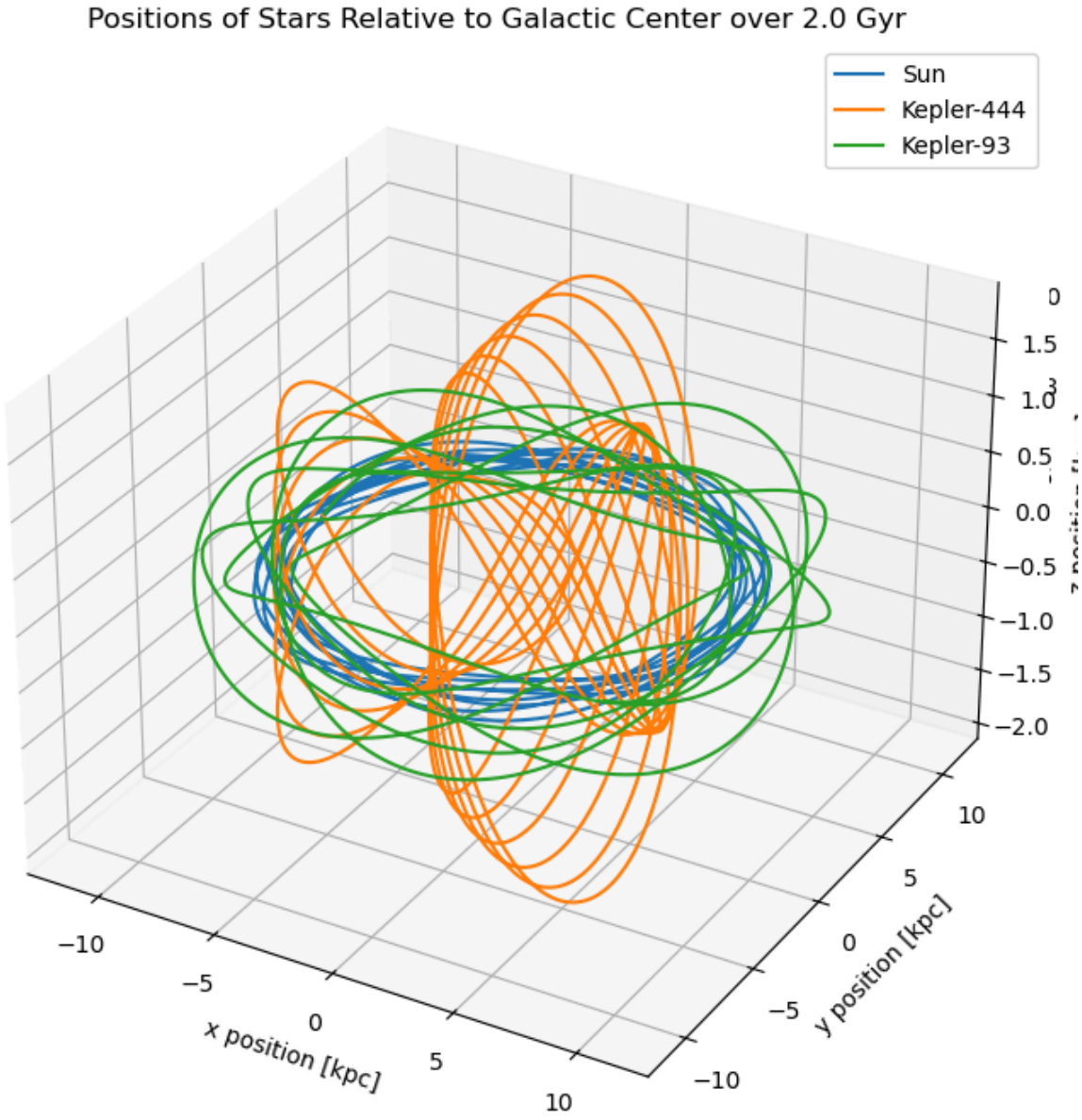


In [11]:
```
# Adding another star_gaia
star_sky = coord.SkyCoord.from_name("Kepler-93")
# Run the sky position cros match
star = GaiaData.from_query(
    f"""
    SELECT TOP 1 * FROM gaiadr3.gaia_source
    WHERE 1=CONTAINS(
        POINT('ICRS', {star_sky.ra.degree}, {star_sky.dec.degree}),
        CIRCLE('ICRS', ra, dec, {(15*u.arcsec).to_value(u.degree)})
    )
    ORDER BY phot_g_mean_mag
    """
)
# Return radial velocity measurements into SkyCoord object
star_c = star.get_skycoord();
# Convert to galactocentric frame
star_gal = star_c.transform_to(galcen_frame)

print(star_gal)

# Initial conditions, orbit computation
star2_w0 = gd.PhaseSpacePosition(star_gal.data)
star2_orb = milk_pot.integrate_orbit(star2_w0,
                                     t=sun_orb.t, # same time for all
                                    )
```

```
<SkyCoord (Galactocentric: galcen_coord=<ICRS Coordinate: (ra, dec) in deg
    (266.4051, -28.936175)>, galcen_distance=8.122 kpc, galcen_v_sun=(12.9,
245.6, 7.78) km / s, z_sun=20.8 pc, roll=0.0 deg): (x, y, z) in pc
    [(-8091.50570343, 89.38644025, 38.15131694)]
 (v_x, v_y, v_z) in km / s
    [(28.67309048, 266.52246068, 23.02647829)]>
```

In [12]:
```
# 3D plots of the integrated orbits

fig4, ax4 = plt.subplots(subplot_kw={"projection": "3d"}, figsize = (7,7))
ax4 = plt.axes(projection='3d')
```

```
ax4.plot3D(sun_orb.pos.x, sun_orb.pos.y, sun_orb.pos.z, label = "Sun");
ax4.plot3D(star_orbit.pos.x, star_orbit.pos.y, star_orbit.pos.z, label = "Ke
ax4.plot3D(star2_orb.pos.x, star2_orb.pos.y, star2_orb.pos.z, label = "Keple

ax4.set_title(f"Positions of Stars Relative to Galactic Center over {t2}");
ax4.set_xlabel("x position [kpc]")
ax4.set_ylabel("y position [kpc]")
ax4.set_zlabel("z position [kpc]");

ax4.legend();
fig4.tight_layout();
```

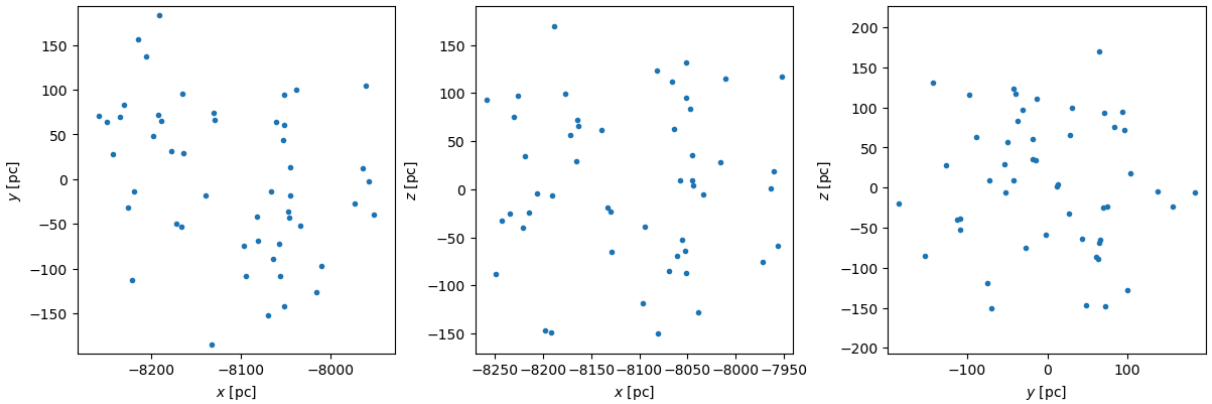


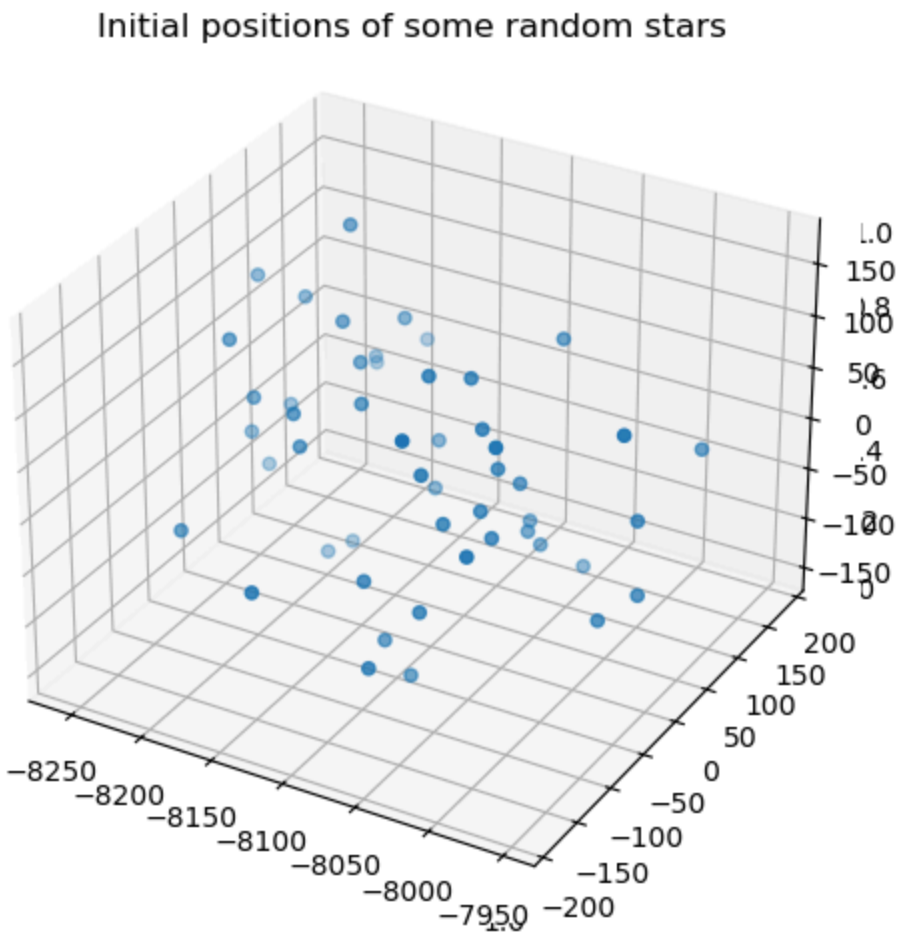
In [13]:
```
# Comparing Orbits to Other Gaia Stars
# Note: this cell takes a long time to run, as it integrates N orbits for N

# Get 50 random Gaia stars within 200 pc of the Sun with well meausred radia
random_stars_g = GaiaData.from_query(
    """
    SELECT TOP 50 * FROM gaiadr3.gaia_source
```

```
    WHERE radial_velocity IS NOT NULL AND
        parallax_over_error > 10 AND
        ruwe < 1.2 AND
        parallax > 5
    ORDER BY random_index
    """
)

# Convert measurements from Gaia into SkyCoord Astropy Object
random_stars_c = random_stars_g.get_skycoord()

# Galactocentric frame
random_stars_galcen = random_stars_c.transform_to(galcen_frame)

# Initial conditions
random_stars_w0 = gd.PhaseSpacePosition(random_stars_galcen.data)

# Integrate the orbits in Milky Way Potential
random_stars_orbits = milk_pot.integrate_orbit(random_stars_w0, t=sun_orb.t)
```

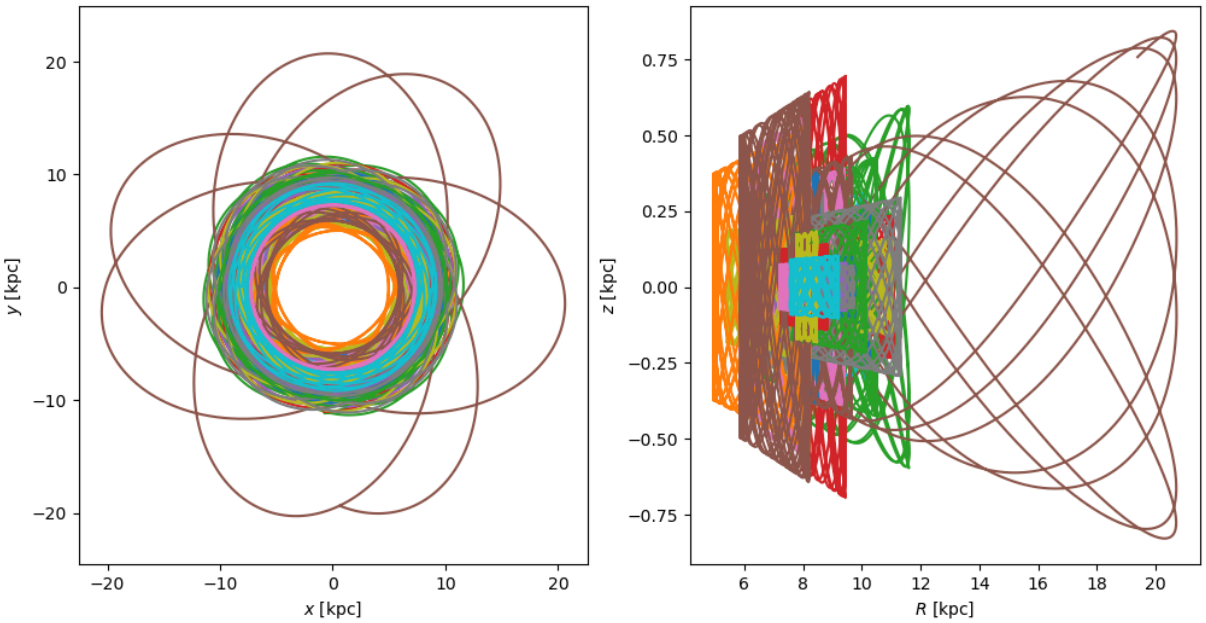
In [14]:
```
# Plot the Initial Positions of the randomly chosen stars

fig5 = random_stars_w0.plot()

x = random_stars_w0.pos.x
y = random_stars_w0.pos.y
z = random_stars_w0.pos.z

fig6, ax6 = plt.subplots(subplot_kw={"projection": "3d"})
ax6 = plt.axes(projection='3d')
ax6.scatter(x, y, z);
fig6.tight_layout();
fig6.suptitle("Initial positions of some random stars")
```

Out[14]: Text(0.5, 0.98, 'Initial positions of some random stars')

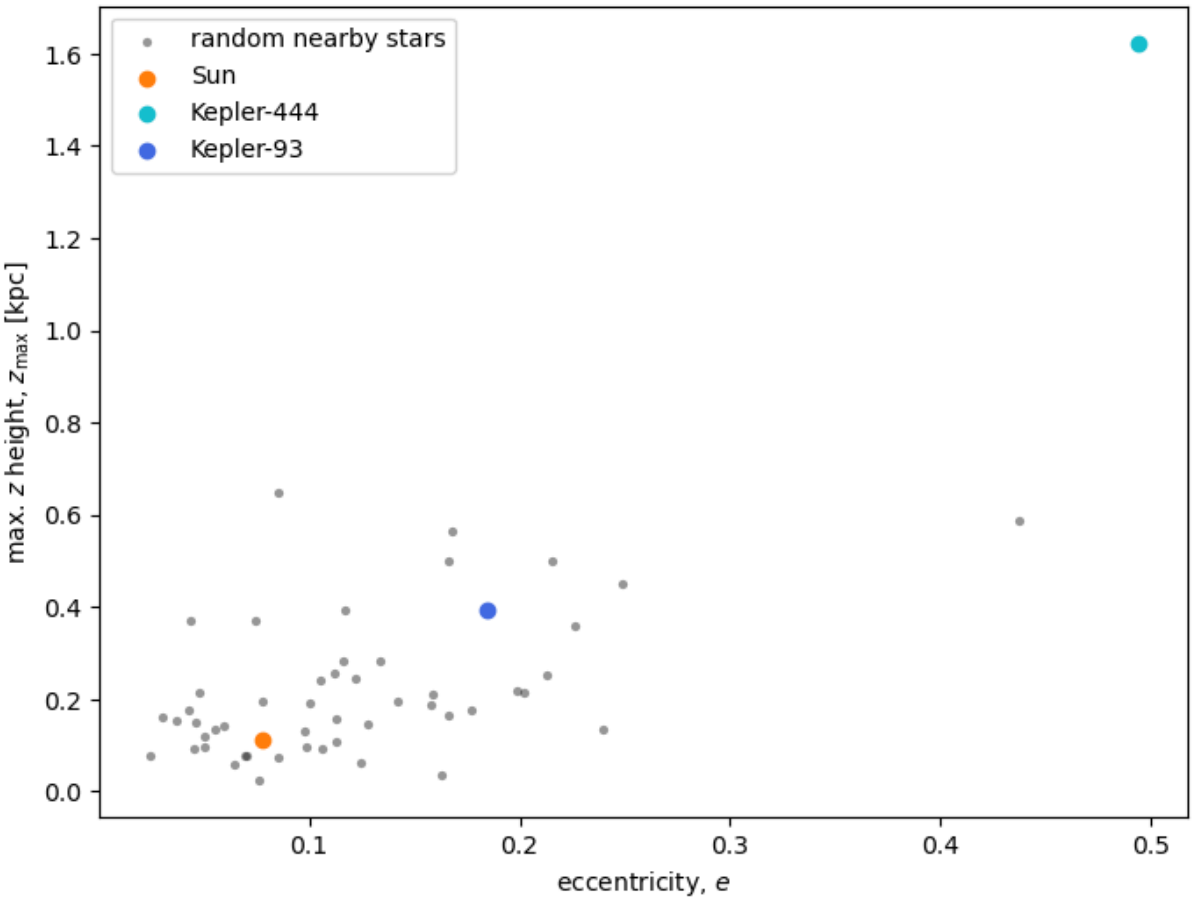




In [15]:
```
# Plot the orbits of random stars

fig7, ax7 = plt.subplots(1, 2, figsize=(12, 6))

random_stars_orbits.plot(["x", "y"], axes=ax7[0])
random_stars_orbits.cylindrical.plot(
    ["rho", "z"],
    axes=ax7[1],
    labels=["$R$ [kpc]", "$z$ [kpc]"],
)
ax7[1].set_aspect("auto")
```



In [16]:
```
# Compare heights and orbits for the selection of stars:

rand_zmax = random_stars_orbits.zmax()
rand_ecc = random_stars_orbits.eccentricity()

fig8, ax8 = plt.subplots(figsize=(8, 6))
ax8.scatter(
    rand_ecc, rand_zmax, color="k", alpha=0.4, s=14, lw=0, label="random nea
)
ax8.scatter(sun_orb.eccentricity(), sun_orb.zmax(), color="tab:orange", labe
ax8.scatter(
    star_orbit.eccentricity(), star_orbit.zmax(), color="tab:cyan", label="K
)
ax8.scatter(
    star2_orb.eccentricity(), star2_orb.zmax(), color="royalblue", label="Ke
)
ax8.legend();
ax8.set_xlabel("eccentricity, $e$")
ax8.set_ylabel(r"max. $z$ height, $z_{\rm max}$ [kpc]");
```



13

The package is not pure python. The package uses C in much of its source code, which is wrapped by Python.

14 - Input

Gala's main inputs include the initial conditions (position, velocity, momentum, etc.) of an object, the gravitational potential surrounding the object, and the time span.

Gala does not generate this data from scratch on its own, but you can randomly select pieces of data from a database. You can also make up your own data and input it for simulation/testing.

15 - Output

Gala generally outputs an "orbit" object for a given celestial body, given the potential it is in and given a particular reference frame, which includes data about position, veloicty, etc.

Programming-wise, Gala outputs an object which includes data.

16 - Unit tests, Regression, Benchmarking

Gala uses unit tests and regression tests, which can be found on the Gala github page. Gala does not appear to have in-built benchmarking.

17 - Reliability

Gala can be a bit unreliable because many of its functions require other packages or data to install, which all have their own updates/version history. This can possibly result in different outputs for the same code given the version of each package on the computer.

18 - Required packages

Gala requires Astropy, numpy, and usually scipy. To make plots, it also normally uses matplotlib as well. Gala's official website says this on the "Getting Started" page.

19 - Documentation

The package provides documentation through its official website and through its GitHub page. The documentation was good for providing code samples to run and see how the different functions of Gala can all fit together. However, the documentation assumes a lot of astronomy and physics knowledge, so it was difficult to parse.

20 - Citations

Gala gives a preferred citation method which is detailed on the website

The creators ask citations for both the paper:

@article{gala, doi = {10.21105/joss.00388}, url = {https://doi.org/10.21105%2Fjoss.00388}, year = 2017, month = {oct}, publisher = {The Open Journal}, volume = {2}, number = {18}, author = {Adrian M. Price-Whelan}, title = {Gala: A Python package for galactic dynamics}, journal = {The Journal of Open Source Software}}

And the Zenodo DOI of the specific version used:

@software{adrian_price_whelan_2020_4159870, author = {Adrian Price-Whelan and Brigitta Sipőcz and Daniel Lenz and Johnny Greco and Nathaniel Starkman and Dan Foreman-Mackey and P. L. Lim and Semyeong Oh and Sergey Koposov and Syrtis Major}, title = {adrn/gala: v1.3}, month = oct, year = 2020, publisher = {Zenodo}, version = {v1.3}, doi = {10.5281/zenodo.4159870}, url = {https://doi.org/10.5281/zenodo.4159870}, }

21 - References

Gala's official website was the main reference used for this report:

https://gala.adrian.pw/en/latest/index.html

Gala's GitHub page was also referenced:

https://github.com/adrn/gala/tree/main

22 - Papers that used this package

@ARTICLE{2025arXiv250509243Z, author = {{Zhang}, Lan and {Xue}, Xiang-Xiang and {Zhu}, Ling and {Zhang}, Ruizhi and {Yang}, Chengqun and {Shao}, Shi and {Chang}, Jiang and {Wang}, Feilu and {Tian}, Hao and {Zhao}, Gang and {Liu}, Chao}, title = "{The Shape and Mass of the Galactic Dark Matter Halo from the Axisymmetric Jeans Model}", journal = {arXiv e-prints}, keywords = {Astrophysics of Galaxies}, year = 2025, month = may, eid = {arXiv:2505.09243}, pages = {arXiv:2505.09243}, abstract = "{We explore the density profile, shape, and virial mass of the Milky Way's dark matter halo using K giants (KG) from LAMOST and SDSS/SEGUE, as well as blue horizontal branch (BHB) stars from SDSS. Incorporating Gaia DR3 proper motions, we first investigate the velocity ellipsoid distribution within the $(R, |z|)$ space. The ellipsoids projected onto the (v_R, v_z) plane exhibit near-spherical alignment. We then probe the underlying dark matter distribution using the axisymmetric Jeans equations with multi-Gaussian expansion (MGE) and the spherically aligned Jeans anisotropic modelling (JAMsph), allowing for different flattened dark matter density models. For each model, we apply two fitting approaches: fitting the KGs and BHBs separately or fit them simultaneously as two dynamical tracers in one gravitational potential. We find consistent results on the dark matter density profiles, r_{200}, and M_{200} within a 1-σ confidence region for models constrained by KGs, BHBs, and both. We find the strongest consistency between KGs and BHBs in constraining dark matter profiles for models incorporating radially varying halo flattening ($q(r_{\rm gc})$), which suggests the Milky Way's dark matter halo shape evolves with Galactocentric distance ($r_{\rm gc}$). Specifically, the halo flattening parameter q_h decreases within $r_{\rm gc} < 20$ kpc and increases for $r_{\rm gc} > 20$ kpc. In this model, $M_{\rm tot}(< 60 \text{ kpc}) = 0.533^{+0.061}_{-0.054} \times 10^{12}\ M_\odot$, r_{200} is 188 ± 15 kpc, with M_{200} estimated at $0.820^{+0.210}_{-0.186} \times 10^{12} M_\odot$.}", doi = {10.48550/arXiv.2505.09243}, archivePrefix = {arXiv}, eprint = {2505.09243}, primaryClass = {astro-ph.GA}, adsurl = {https://ui.adsabs.harvard.edu/abs/2025arXiv250509243Z}, adsnote = {Provided by the SAO/NASA Astrophysics Data System} }@ARTICLE{2025ApJ...982L..37C, author = {{Cao}, Chunyang and {Liu}, F.~K. and {Li}, Shuo and {Chen}, Xian and {Wang}, Ke}, title = "{A Recent Supermassive Black Hole Binary in the Galactic Center Unveiled by the Hypervelocity Stars}", journal = {\apjl}, keywords = {Galactic center, Hypervelocity stars, Supermassive black holes, Galaxy mergers, Binary stars, 565, 776, 1663, 608, 154, High Energy Astrophysical Phenomena, Astrophysics of Galaxies}, year = 2025, month = apr, volume = {982}, number = {2}, eid = {L37}, pages = {L37}, doi = {10.3847/2041-8213/adbbf2}, archivePrefix = {arXiv}, eprint = {2411.09278}, primaryClass = {astro-ph.HE}, adsurl = {https://ui.adsabs.harvard.edu/abs/2025ApJ...982L..37C}, adsnote = {Provided by the SAO/NASA Astrophysics Data System} }

23 - New Python Methods

To use Gala, I had to gain familiarity with a couple things not covered in this course, the main one being classes in Python.

24 - Prior Experience

I had no prior experience in this package.

I did not collaborate in a group.