Practicas 'Topicos de Astronomia moderna - 2024A'

Mary Verdugo

A continuacion se desarrollara el codigo y las preguntas correspondiente a las practicas propuestas por el profesor y discutidas en clases con los demas companeros. Se obviara la instalacion de los paquetes y tutorial (En mi caso, como usuario de MacOS fue necesario actualizar las librerias de anaconda y descargar la version de desarrollador de GALA, http://gala.adrian.pw/en/latest/)

Referencias:

- https://doi.org/10.5281/zenodo.4159870
- https://doi.org/10.21105%2Fjoss.00388

Practica I

Objetivo: Explorar el comportamiento de orbitas estelares en distintos potenciales galacticos

1. Que componentes principales debemos elegir para una galaxia tipo L* tardio?

Hacemos un **potencial compuesto** usando la funcion de gala `gp.CCompositePotential`. El potencial de dicha galaxia se puede dividir en:

- Disk: Miyamoto-Nagai
- Bulge: Hernquist
- Dark Matter halo: Navarro Frenk and White (en primera instancia usaremos el esferico, no triaxial)

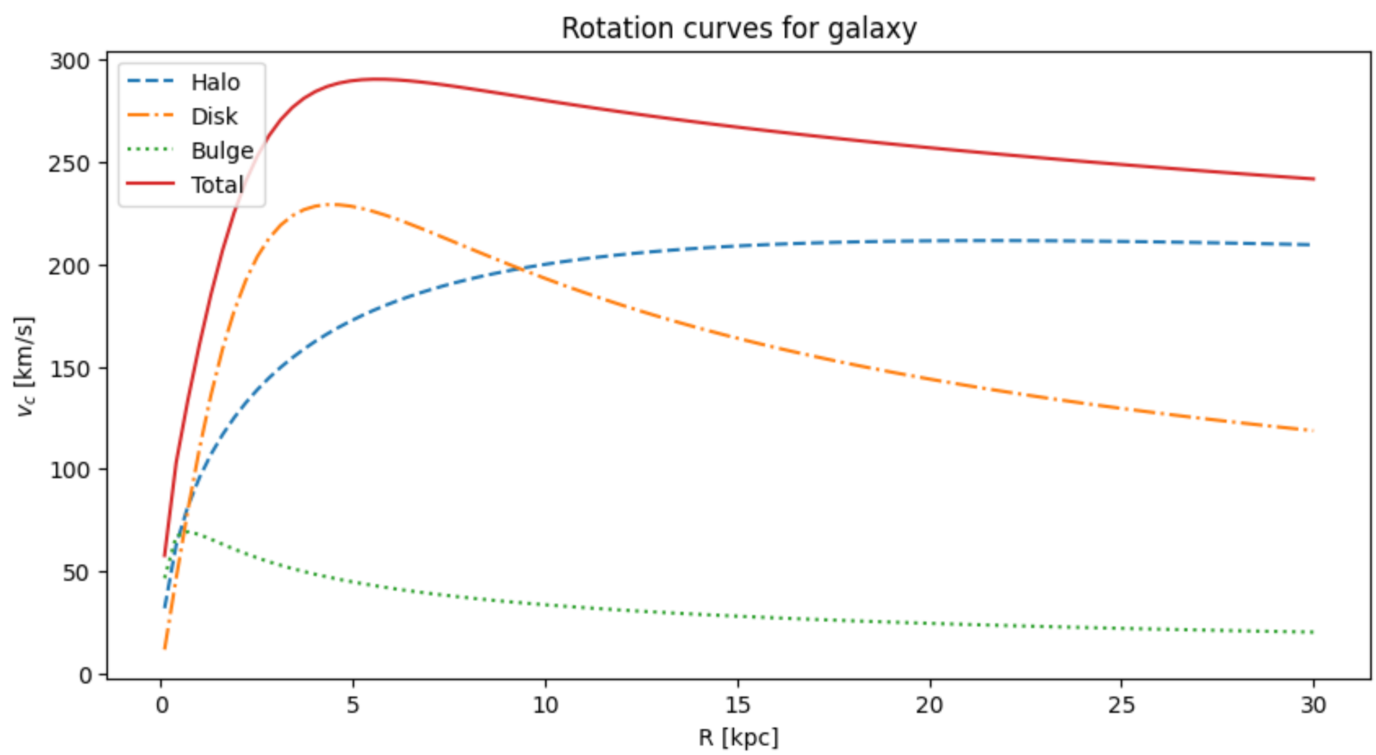
```
#We import the libraries and the differents tools of
#pyhton to make figures and units from astropy:
#python version 3.12.2
import gala.potential as gp
import astropy.units as u
import numpy as np
import gala.dynamics as gd
from gala.units import galactic
import matplotlib.pyplot as plt
```

```
total_potential = gp.CCompositePotential()
total_potential['disk'] = gp.MiyamotoNagaiPotential(m = 1E11 , a=3, b=0.15, un
total_potential['bulge'] = gp.HernquistPotential(m = 3E9 , c = 0.67, units=gal
total_potential['dm_halo'] = gp.NFWPotential.from_circular_velocity(v_c=200*u.
                                                 r_s=10.*u.kpc,
                                                 units=galactic)
```

Analogo al tutorial que es para una galaxia tipo MW, (https://gala.adrian.pw/en/latest/tutorials.html), lo completamos para el potencial de nuestra galaxia L* tardio (`total_potential`).

2. Calcular las curvas de velocidades para:
    a. Cada componente
    b. Sistema total

```python
R_grid = np.linspace(0.1, 30, 100) * u.kpc
xyz = np.zeros((3,) + R_grid.shape) * total_potential['dm_halo'].units["length
xyz[0] = R_grid

vcirc_halo = total_potential['dm_halo'].circular_velocity(xyz)
vcirc_disk = total_potential['disk'].circular_velocity(xyz)
vcirc_bulge = total_potential['bulge'].circular_velocity(xyz)
vcirc_gal = total_potential.circular_velocity(xyz)

fig = plt.figure(figsize=(10, 5))
plt.plot(R_grid, vcirc_halo, label='Halo',linestyle='--')
plt.plot(R_grid, vcirc_disk, label='Disk',linestyle='-.')
plt.plot(R_grid, vcirc_bulge, label='Bulge',linestyle=':')
plt.plot(R_grid, vcirc_gal, label='Total')
plt.xlabel('R [kpc]')
plt.ylabel(f'$v_c$ [km/s]')
plt.title('Rotation curves for galaxy')
plt.legend()
plt.show()
```



3. Usando solo el potencial esferico de DM halo lograr:
    a. Orbita cerrada (E < 0 [km/s^{2}])
    b. Orbita hiperbolica (E > 0 [km/s^{2}])

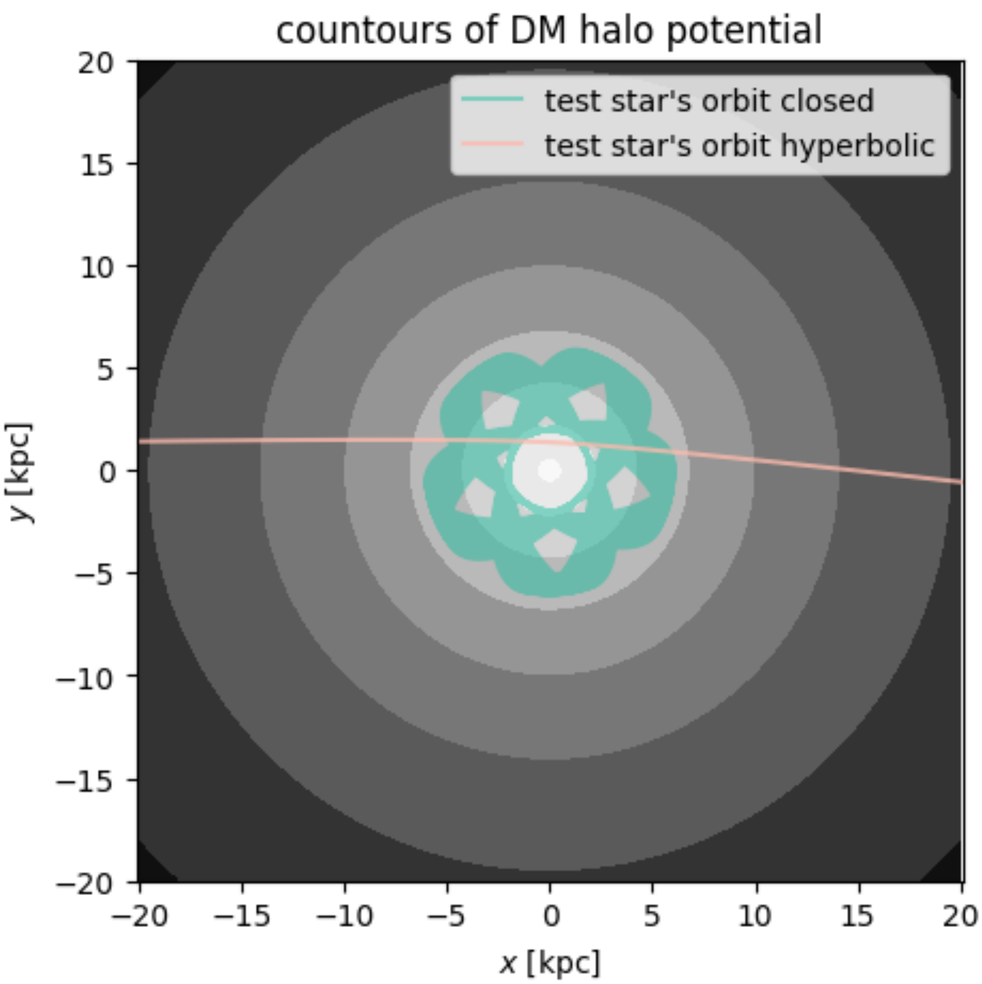
Calcularemos la orbita de las siguientes estrellas de prueba (`a y b`), usando solo el potencial del DM halo (`total_potential['dm_halo']`).

```
a_star_p = [6,0,0]
a_star_v = [20,80,0]

b_star_p = [-100,0,0]
b_star_v = [500,10,0]

a_particle = gd.PhaseSpacePosition(pos = a_star_p*u.kpc,
                            vel = a_star_v*u.km/u.s)
b_particle = gd.PhaseSpacePosition(pos = b_star_p*u.kpc,
                                   vel= b_star_v*u.km/u.s)
```

```
# Usando el hamiltoniano calculamos la energia de la orbita,
# dadas las condiciones iniciales que definimos en la celda anterior
# esto para corroborar que tipo de orbita es.
a_orbit_DM = gp.Hamiltonian(total_potential['dm_halo']).integrate_orbit(a_part
                                                                        dt=0.5
                                                                        t1=0,
b_orbit_DM = gp.Hamiltonian(total_potential['dm_halo']).integrate_orbit(b_part
                                                                        dt =0.
                                                                        t1=0,

```

```python
grid = np.linspace(-20,20,128)
fig, ax = plt.subplots(1,1, figsize=(5,5))
fig = total_potential['dm_halo'].plot_contours(grid=(grid, grid, 0), cmap='Gre
fig = a_orbit_DM.plot(['x','y'], color='#1ABC9C',
                      alpha=0.5, axes=[ax],
                      auto_aspect=True,
                      label="test star's orbit closed")
fig2 = b_orbit_DM.plot(['x','y'], color='#FCBBAE',
                       alpha=0.8, axes=[ax],
                       auto_aspect=True,
                       label="test star's orbit hyperbolic")
ax.set_xlim(-20,20)
ax.set_ylim(-20,20)

ax.legend()
ax.set_title('countours of DM halo potential')

```

```
Text(0.5, 1.0, 'countours of DM halo potential')
```



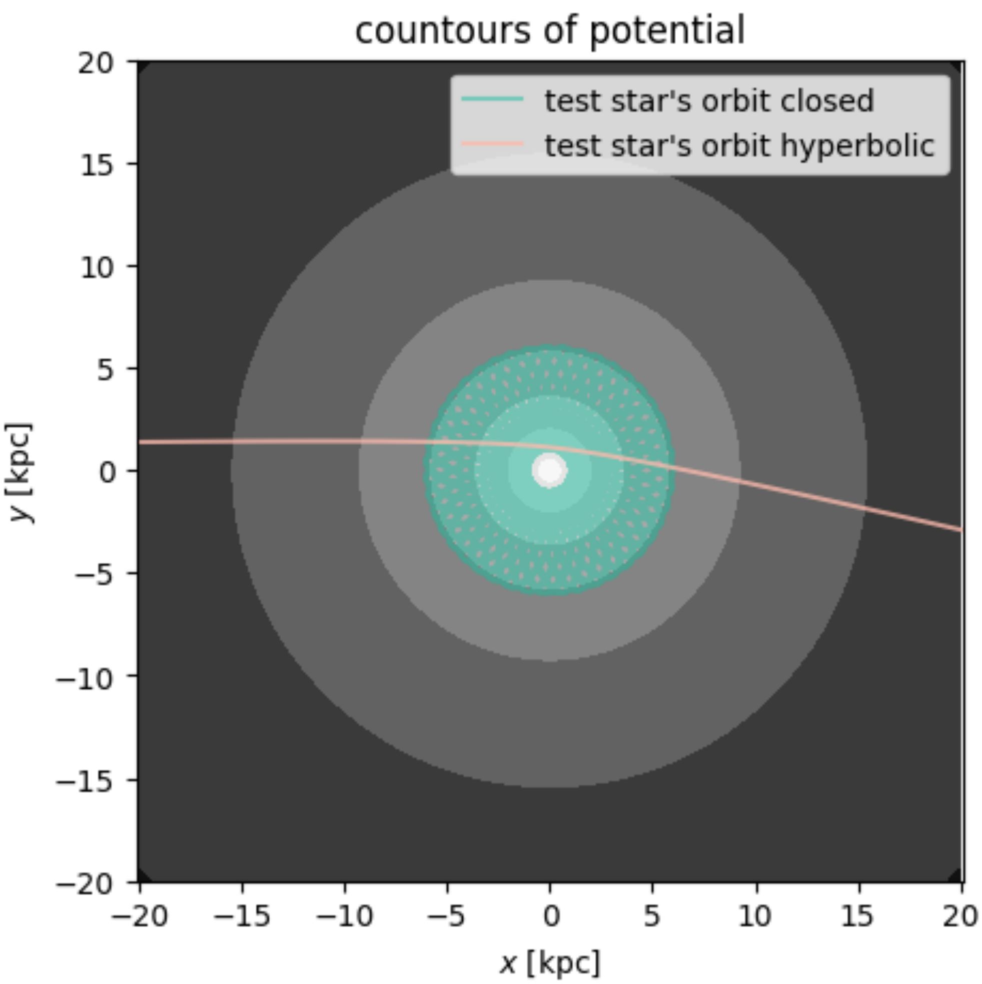
4. Integrar orbitas anteriores e ir agregando componentes.

    - DM halo (`total_potential['dm_halo']`)
    - DM halo + disco (`total_potential['dm_halo'] + total_potential['disk']`)
    - DM halo + disco + bulge (`total_potential['dm_halo'] + total_potential['disk'] + total_potential['bulge']`)

```
a_orbit = gp.Hamiltonian(total_potential).integrate_orbit(a_particle, dt=0.5*u
                                                                               t1=0, t2=1
b_orbit = gp.Hamiltonian(total_potential).integrate_orbit(b_particle,dt =0.5*u
                                                                               t1=0, t2=1

```

```
grid = np.linspace(-20,20,128)
fig, ax = plt.subplots(1,1, figsize=(5,5))
fig = total_potential.plot_contours(grid=(grid, grid, 0),
                                    cmap='Greys', ax=ax)
fig = a_orbit.plot(['x','y'], color='#1ABC9C',
                      alpha=0.5, axes=[ax],
                      auto_aspect=True,
                      label="test star's orbit closed")
fig2 = b_orbit.plot(['x','y'], color='#FCBBAE',
                      alpha=0.8, axes=[ax],
                      auto_aspect=True,
                      label="test star's orbit hyperbolic")
ax.set_xlim(-20,20)
ax.set_ylim(-20,20)

ax.legend()
ax.set_title('countours of potential')
```

```
Text(0.5, 1.0, 'countours of potential')
```

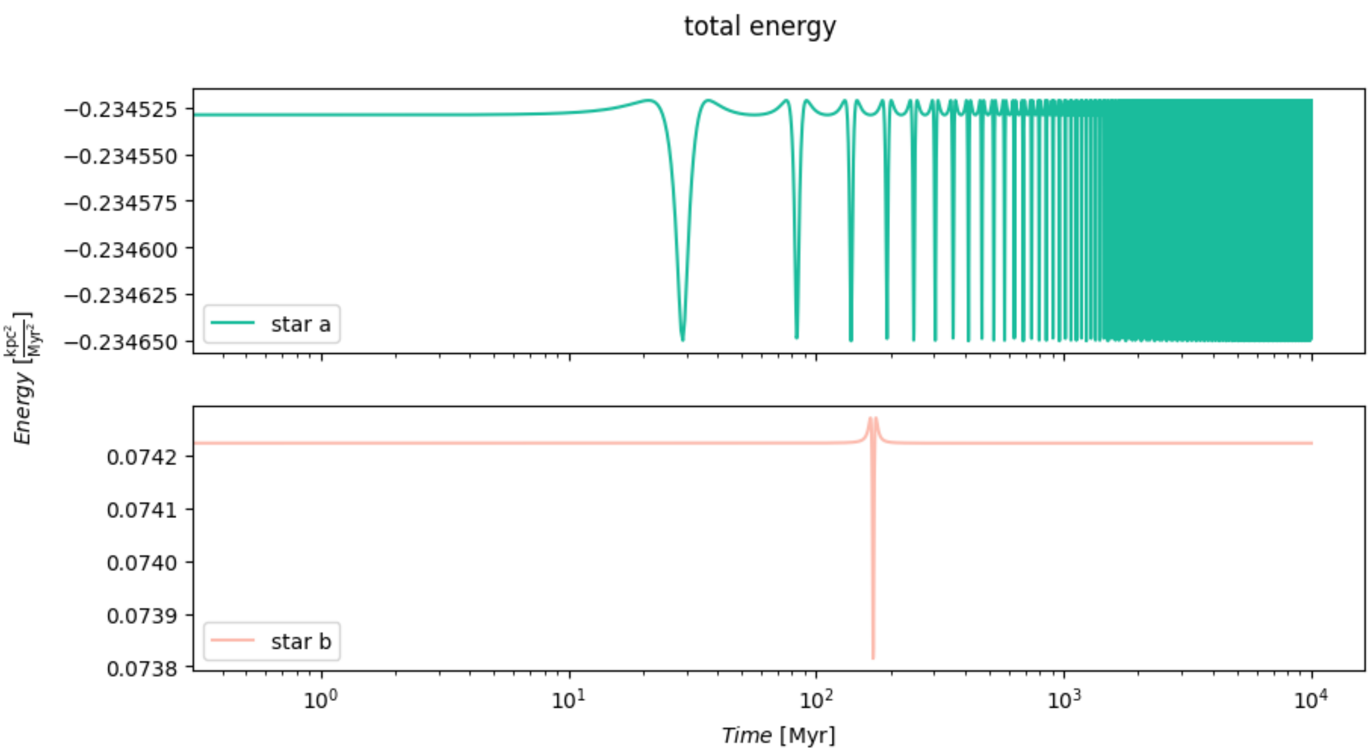


Podemos notar que el espacio se hace denso mucho mas rapido que cuando solo trabajamos sobre el halo esferico de materia oscura. Por otro lado la orbita hiperbolica, sigue siendo hiperbolica.
Tambien graficaremos la energia de la particula en el tiempo, y comprobaremos si efectivamente son cerrada e hiperbolica respectivamente.

```
b_totalenergy = b_orbit.energy().to(u.kpc**2/u.Myr**2)

a_totalenergy = a_orbit.energy().to(u.kpc**2/u.Myr**2)
```

```python
fig, ax = plt.subplots(2,1, figsize=(10,5), sharex=True)

ax[0].plot(a_orbit.t, a_totalenergy,
           color='#1ABC9C', label='star a')
ax[1].plot(b_orbit.t, b_totalenergy,
         color='#FCBBAE', label='star b')

ax[1].set_xlabel("$Time$ [{}]".format(a_orbit.t.unit.to_string(format="latex")
ax[0].set_xscale('log')
ax[1].set_xscale('log')

fig.text(0.03, 0.5, "$Energy$ [{}]".format(a_totalenergy.unit.to_string(format
        horizontalalignment='right',
        verticalalignment='center',
        rotation='vertical')
plt.suptitle('total energy')
ax[0].legend(loc='best')
ax[1].legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x1485df4d0>
```



Como podemos notar, la energia total es negativa en el caso de la estrella a, es decir, esta cerrada, mientras que para la estrella b (cabe destacar que su posicion es mucho mas lejana a la galaxia que la posicion inicial de a), la energia total se mantiene siempre positiva.
Podemos observar tambien que la evolucion de la energia en el tiempo nos indica que tan lejos o cerca se encuentra el cuerpo del area de mayor potencial de la galaxia. Si consideramos que la energia total es la suma de la energia cintica y potencial, notamos que estas se mantienen en equilibrio para la orbita cerrada, subiendo y bajando de manera complementaria, mientras que para la orbita hiperbolica, la energia potencial baja cuando se acerca a la galaxia, pero no es suficiente para que la energia total sea negativa, es decir la energia cinetica no disminuye y por ende la energia total es mayor a 0.

```
a_eccentricity = a_orbit.eccentricity()
b_eccentricity = b_orbit.eccentricity()

a_apocenter = a_orbit.apocenter()
b_apocenter = b_orbit.apocenter()

a_pericenter = a_orbit.pericenter()
b_pericenter = b_orbit.pericenter()
```

```
/opt/anaconda3/envs/astro/lib/python3.12/site-packages/numpy/core/fromnumeric.
  return _methods._mean(a, axis=axis, dtype=dtype,
/opt/anaconda3/envs/astro/lib/python3.12/site-packages/numpy/core/_methods.py
  ret = ret.dtype.type(ret / rcount)
```

Gala utiliza la ecuacion: $e = \frac{r_{\text{apo}} - r_{\text{per}}}{r_{\text{apo}} + r_{\text{per}}}$ Y con ello podemos obtener los valores correspondientes a la eccentiricidad. Mientras que el apocentro y pericentro promedio provienen de la integracion numerica de la orbita

```

print(a_eccentricity, a_apocenter, a_pericenter)
print(b_eccentricity,b_apocenter, b_pericenter )
```

```
0.7095922426112573 6.01542239391078 kpc 1.021837420421007 kpc
nan nan kpc 1.1152944507957727 kpc
```

star	eccentricidad	apocentro	pericentro
a star	0.7095922426112573	6.015 kpc	1.022 kpc
b star	nan	nan	1.12 kpc

6. Comparar y discutir:

Podemos notar que la eccentricidad de la orbita de a es menor a 1, que por definicion significa que es una orbita eliptica. Luego tenemos que su apocentro medio es ~~6 kpc y su pericentro ~1kpc. Considerando que la estrella b parte por fuera de la galaxia y no queda ligada, notamos que no hay un valor para la eccentricidad y el apocentro, que deben ser~~ $e_b >> 1$ ~~y apocentro~~ $\rightarrow \infty$~~, pero que su pericentro es similar al de a~~ (1kpc). Esto podria deberse a que ambos se acercan a la galaxia en el mismo plano (es decir con posicion inicia en z iguales)

Practica II

Objetivo: Seguimiento a la practica anterior para explorar mas propiedades aprendidas en clases.

1. Considerar orbitas cerradas solamente. De forma secuencial graficar:
   - Energia vs tiempo
   - momento angular total vs tiempo
   - componentes de momento angular vs tiempo
   - eccentricidad vs tiempo

```
def calculate_eccentricity(apocenter, pericenter):
    eccentricity = (apocenter - pericenter) / (apocenter + pericenter)
    return eccentricity
```

```
# Respecto a la practica anterior utilizaremos la estrella a
# que es cerrada, y calcularemos su momento angular

#para el dm halo
aDM_totalenergy = a_orbit_DM.energy()

aDM_angularmoment = a_orbit_DM.angular_momentum
aDM_Lx = aDM_angularmoment()[0]
aDM_Ly = aDM_angularmoment()[1]
aDM_Lz = aDM_angularmoment()[2]
aDM_totalangularmoment = np.sqrt(aDM_Lx**2 +
                               aDM_Ly**2 +
                               aDM_Lz**2)
aDM_pericenter = (a_orbit_DM.pericenter(return_times=True, func=None)[0]).to_v
aDM_apocenter = (a_orbit_DM.apocenter(return_times=True, func=None)[0]).to_val
```

```
aDM_apocenter = np.delete(aDM_apocenter, -1)
timeDMecc = a_orbit_DM.pericenter(return_times=True, func=None)[1]
aDM_ecc = calculate_eccentricity(aDM_apocenter, aDM_pericenter)

#para el disco
a_orbit_disk = gp.Hamiltonian(total_potential['disk']).integrate_orbit(a_parti
                                                                   dt=0.5*u.Myr
                                                                     t1=0, t2=1
a_disk_totalenergy = a_orbit_disk.energy()

a_disk_angularmoment = a_orbit_disk.angular_momentum
a_disk_Lx = a_disk_angularmoment()[0]
a_disk_Ly = a_disk_angularmoment()[1]
a_disk_Lz = a_disk_angularmoment()[2]
a_disk_totalangularmoment = np.sqrt(a_disk_Lx**2 +
                               a_disk_Ly**2 +
                               a_disk_Lz**2)
a_disk_pericenter = (a_orbit_disk.pericenter(return_times=True, func=None)[0])
a_disk_apocenter = (a_orbit_disk.apocenter(return_times=True, func=None)[0]).t
a_disk_apocenter = np.delete(a_disk_apocenter, -1)
timediskecc = a_orbit_disk.pericenter(return_times=True, func=None)[1]
a_disk_ecc = calculate_eccentricity(a_disk_apocenter, a_disk_pericenter)



#para el bulbo
a_orbit_bulge = gp.Hamiltonian(total_potential['bulge']).integrate_orbit(a_par
                                                                   dt=0.5*u.Myr
                                                                     t1=0, t2=1
a_bulge_totalenergy = a_orbit_bulge.energy()

a_bulge_angularmoment = a_orbit_bulge.angular_momentum
a_bulge_Lx = a_bulge_angularmoment()[0]
a_bulge_Ly = a_bulge_angularmoment()[1]
a_bulge_Lz = a_bulge_angularmoment()[2]
a_bulge_totalangularmoment = np.sqrt(a_bulge_Lx**2 +
                               a_bulge_Ly**2 +
                               a_bulge_Lz**2)

a_bulge_pericenter = (a_orbit_bulge.pericenter(return_times=True, func=None)[0
a_bulge_apocenter = (a_orbit_bulge.apocenter(return_times=True, func=None)[0])
#a_bulge_apocenter = np.delete(a_bulge_apocenter, -1)
timebulgeecc = a_orbit_bulge.pericenter(return_times=True, func=None)[1]
a_bulge_ecc = calculate_eccentricity(a_bulge_apocenter, a_bulge_pericenter)


#para todos los potenciales
a_angularmoment = a_orbit.angular_momentum
```

```
a_Lx = a_angularmoment()[0]
a_Ly = a_angularmoment()[1]
a_Lz = a_angularmoment()[2]
a_totalangularmoment = np.sqrt(a_Lx**2 +
                               a_Ly**2 +
                               a_Lz**2)

timeecc = a_orbit.pericenter(return_times=True, func=None)[1]
a_apocenter = (a_orbit.apocenter(return_times=True, func=None)[0]).to_value()
a_pericenter = (a_orbit.pericenter(return_times=True, func=None)[0]).to_value(
a_apocenter = np.delete(a_apocenter, -1)
a_ecc = calculate_eccentricity(a_apocenter, a_pericenter)
```

```
from mpl_toolkits.axes_grid1.inset_locator import zoomed_inset_axes
from mpl_toolkits.axes_grid1.inset_locator import mark_inset
```

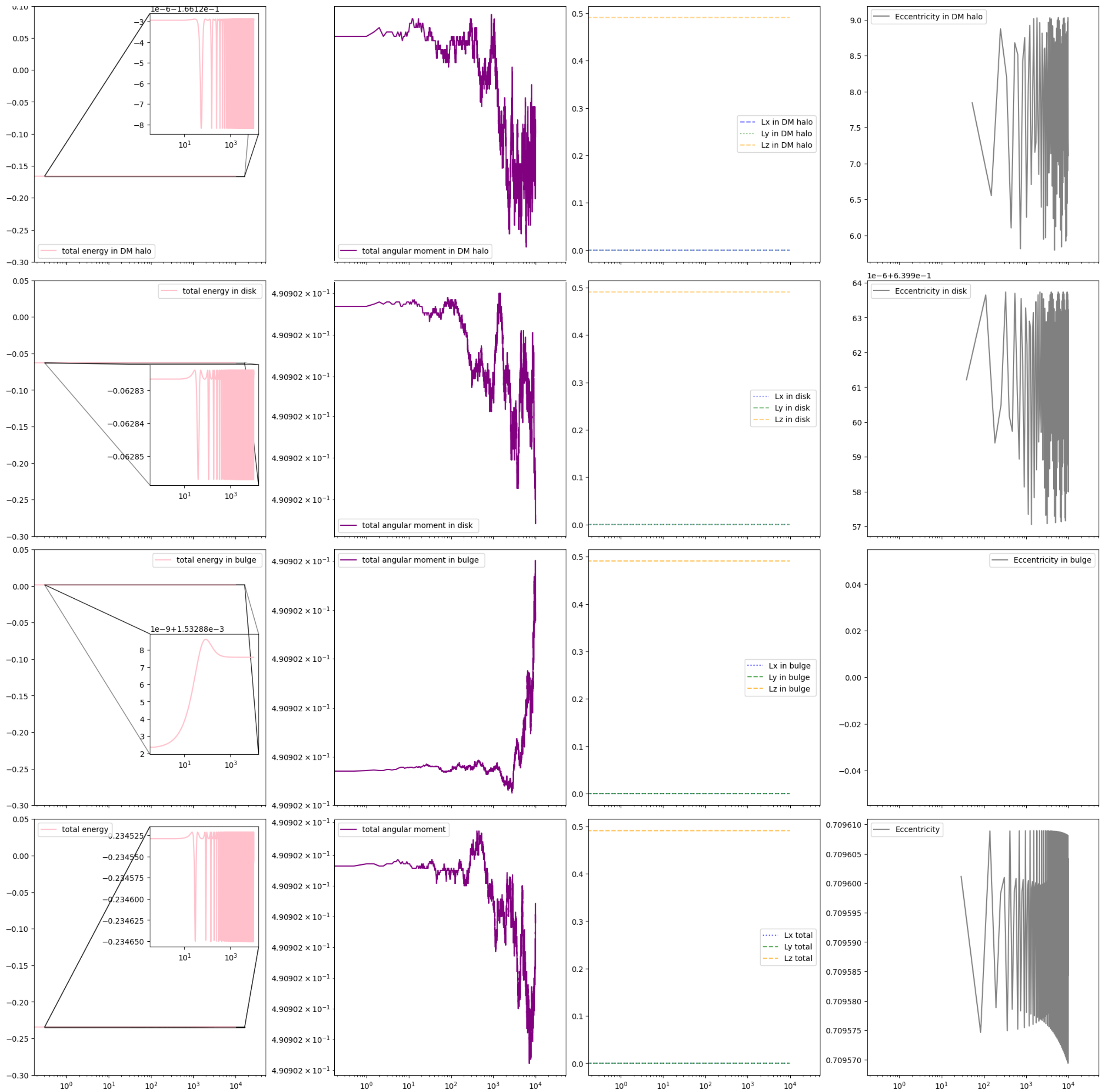
```
t = a_orbit.t


fig, ax = plt.subplots(4,4, figsize=(20,20), constrained_layout = True, sharex
ax[0,0].set_xscale('log')
ax[0,0].plot(t, aDM_totalenergy, color='pink', label='total energy in DM halo'
ax[0,0].legend(loc='best')
ax[0,0].set_ylim(-0.3,0.1)

axins = ax[0,0].inset_axes([0.5, 0.5, 0.47, 0.47])
# axins = zoomed_inset_axes(ax[0,0], 3, loc=4) # zoom = 6
# sub region of the original image

axins.set_xscale('log')
axins.plot(t, aDM_totalenergy, color='pink', label='total energy in DM halo')

# draw a bbox of the region of the inset axes in the parent axes and
# connecting lines between the bbox and the inset axes area
ax[0,0].indicate_inset_zoom(axins, edgecolor="black")
mark_inset(ax[0,0], axins, loc1=2, loc2=4, fc='none',ec='0.1')




ax[0,1].plot(t, aDM_totalangularmoment, color='purple', label='total angular m
ax[0,1].legend(loc='best')
ax[0,1].set_yscale('log')
```

```
ax[0,2].plot(t, aDM_Lx, color='blue', alpha=0.5,label='Lx in DM halo', linesty
ax[0,2].plot(t, aDM_Ly, color='green', alpha=0.5, label='Ly in DM halo', lines
ax[0,2].plot(t, aDM_Lz, color='orange', alpha=0.5, label='Lz in DM halo', line
ax[0,2].legend(loc='best')

ax[0,3].plot(timeDMecc, aDM_ecc, color='gray', label='Eccentricity in DM halo'
ax[0,3].legend(loc='best')


ax[1,0].set_xscale('log')
ax[1,0].plot(t, a_disk_totalenergy, color='pink', label='total energy in disk
ax[1,0].legend(loc='best')
ax[1,0].set_ylim(-0.3,0.05)

axins1 = ax[1,0].inset_axes([0.5, 0.2, 0.47, 0.47])
# axins = zoomed_inset_axes(ax[0,0], 3, loc=4) # zoom = 6
# sub region of the original image

axins1.set_xscale('log')
axins1.plot(t, a_disk_totalenergy, color='pink')

# draw a bbox of the region of the inset axes in the parent axes and
# connecting lines between the bbox and the inset axes area
ax[1,0].indicate_inset_zoom(axins1, edgecolor="black")
mark_inset(ax[1,0], axins1, loc1=2, loc2=4, fc='none',ec='0.1')

ax[1,1].plot(t, a_disk_totalangularmoment, color='purple', label='total angula
ax[1,1].legend(loc='best')
ax[1,1].set_yscale('log')


ax[1,2].plot(t, a_disk_Lx, color='blue', alpha=0.5,label='Lx in disk ',linesty
ax[1,2].plot(t, a_disk_Ly, color='green', alpha=0.5, label='Ly in disk ',lines
ax[1,2].plot(t, a_disk_Lz, color='orange', alpha=0.5, label='Lz in disk ',line
ax[1,2].legend(loc='best')

ax[1,3].plot(timediskecc, a_disk_ecc, color='gray', label='Eccentricity in dis
ax[1,3].legend(loc='best')



ax[2,0].set_xscale('log')
ax[2,0].plot(t, a_bulge_totalenergy, color='pink', label='total energy in bulg
ax[2,0].legend(loc='best')
ax[2,0].set_ylim(-0.3,0.05)

axins2 = ax[2,0].inset_axes([0.5, 0.2, 0.47, 0.47])
```

```python
axins2.set_xscale('log')
axins2.plot(t, a_bulge_totalenergy, color='pink')

# draw a bbox of the region of the inset axes in the parent axes and
# connecting lines between the bbox and the inset axes area
ax[2,0].indicate_inset_zoom(axins2, edgecolor="black")
mark_inset(ax[2,0], axins2, loc1=2, loc2=4, fc='none',ec='0.1')

ax[2,1].plot(t, a_bulge_totalangularmoment, color='purple', label='total angul
ax[2,1].legend(loc='best')
ax[2,1].set_yscale('log')

ax[2,2].plot(t, a_bulge_Lx, color='blue', alpha=0.7,label='Lx in bulge ',lines
ax[2,2].plot(t, a_bulge_Ly, color='green', alpha=0.7, label='Ly in bulge', lin
ax[2,2].plot(t, a_bulge_Lz, color='orange', alpha=0.7, label='Lz in bulge ',li
ax[2,2].legend(loc='best')

ax[2,3].plot(timebulgeecc, a_bulge_ecc, color='gray', label='Eccentricity in b
ax[2,3].legend(loc='best')



ax[3,0].set_xscale('log')
ax[3,0].plot(t, a_totalenergy, color='pink', label='total energy')
ax[3,0].legend(loc='best')
ax[3,0].set_ylim(-0.3,0.05)
axins3 = ax[3,0].inset_axes([0.5, 0.5, 0.47, 0.47])
axins3.set_xscale('log')
axins3.plot(t, a_totalenergy, color='pink')
# draw a bbox of the region of the inset axes in the parent axes and
# connecting lines between the bbox and the inset axes area
ax[3,0].indicate_inset_zoom(axins3, edgecolor="black")
mark_inset(ax[3,0], axins3, loc1=2, loc2=4, fc='none',ec='0.1')

ax[3,1].plot(t, a_totalangularmoment, color='purple', label='total angular mom
ax[3,1].legend(loc='best')
ax[3,1].set_yscale('log')

ax[3,2].plot(t, a_Lx, color='blue', alpha=0.7,label='Lx total', linestyle=':')
ax[3,2].plot(t, a_Ly, color='green', alpha=0.7, label='Ly total', linestyle='-
ax[3,2].plot(t, a_Lz, color='orange', alpha=0.7, label='Lz total', linestyle='
ax[3,2].legend(loc='best')

ax[3,3].plot(timeecc, a_ecc, color='gray', label='Eccentricity')
ax[3,3].legend(loc='best')

```

```
<matplotlib.legend.Legend at 0x1495d78c0>
```

1e−7+5.12793e−1



total energy in DM halo
total angular moment in DM halo
Lx in DM halo
Ly in DM halo
Lz in DM halo
Eccentricity in DM halo
total energy in disk
total angular moment in disk
Lx in disk
Ly in disk
Lz in disk
Eccentricity in disk
total energy in bulge
total angular moment in bulge
Lx in bulge
Ly in bulge
Lz in bulge
Eccentricity in bulge
total energy
total angular moment
Lx total
Ly total
Lz total
Eccentricity

Notemos que el bulbo no muetra eccentricidad para la estrella a. Esto es porque en la deifnicon delpotencial del bulbo, su masa es demasiado baja para atraer gravitatoriamente a la estrella a y dejarla ligada. Se puede corroborar observando que la orbita que toma la estrella cuando solo entra al bulbo tiene $E > 0$. Y dentro de la segunda columna, podemos ver que enrealidad el ruido que se nota sobre la curva de momento angular es infimo, por ende este es constante en general, teniendo claras variaciones para el caso del bulbo, que no representa necesariamente error debido al metodo numerico de aproximacion (leap-frog, default integrator [https://gala.adrian.pw/en/latest/tutorials/pyia-gala-orbit.html]) y tiene relacion con lo mencionado previamente.

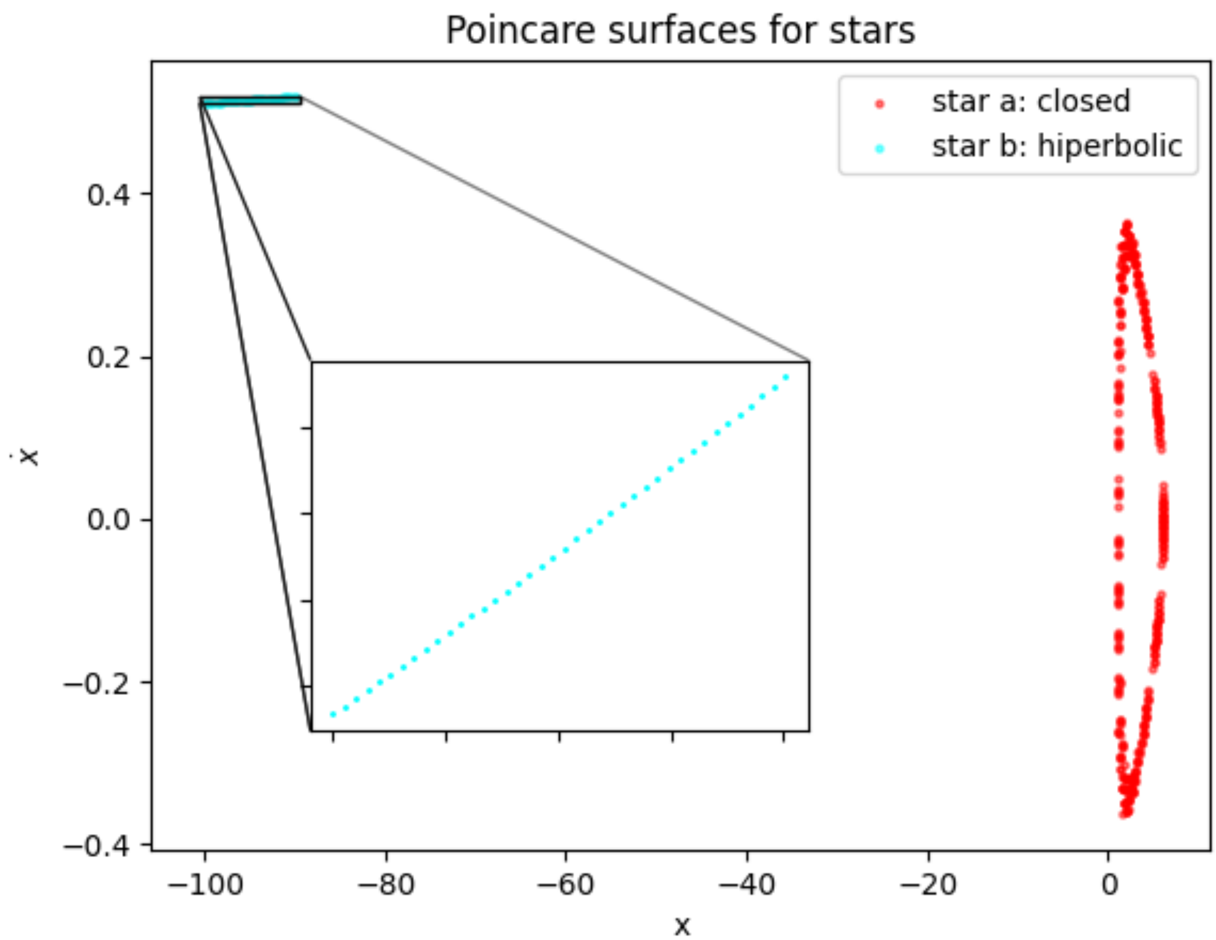
```
position_x_a = a_orbit.x.value
velocity_x_a = a_orbit.vel.d_x.value
position_y_a = a_orbit.y.value
velocity_y_a = a_orbit.vel.d_y.value
position_z_a = a_orbit.z.value
velocity_z_a = a_orbit.vel.d_z.value
delta = 0.2

poincare_index_a = np.where((position_y_a >= -delta) & (position_y_a <= delta)
#poincare_index_c = np.where((position_y_c == 0) & (velocity_y_c > 0))[0]

poincare_position_x_a = position_x_a[poincare_index_a]
poincare_velocity_x_a = velocity_x_a[poincare_index_a]
poincare_velocity_y_a = velocity_y_a[poincare_index_a]
poincare_position_y_a = position_y_a[poincare_index_a]
poincare_position_z_a = position_z_a[poincare_index_a]
poincare_velocity_z_a = velocity_z_a[poincare_index_a]
```

```
position_x_b = b_orbit.x.value
velocity_x_b = b_orbit.vel.d_x.value
position_y_b = b_orbit.y.value
velocity_y_b = b_orbit.vel.d_y.value
position_z_b = b_orbit.z.value
velocity_z_b = b_orbit.vel.d_z.value
delta = 0.2

poincare_index_b = np.where((position_y_b >= -delta) & (position_y_b <= delta)
#poincare_index_b = np.where((position_y_b == 0) & (velocity_y_b > 0))[0]

poincare_position_x_b = position_x_b[poincare_index_b]
poincare_velocity_x_b = velocity_x_b[poincare_index_b]
poincare_velocity_y_b = velocity_y_b[poincare_index_b]
poincare_position_y_b = position_y_b[poincare_index_b]
poincare_position_z_b = position_z_b[poincare_index_b]
poincare_velocity_z_b = velocity_z_b[poincare_index_b]
```

```
fig, ax  = plt.subplots()

ax.scatter(poincare_position_x_a, poincare_velocity_x_a,color='red', alpha=0.5
ax.scatter(poincare_position_x_b, poincare_velocity_x_b, color='cyan', alpha=0

axins = ax.inset_axes([0.15, 0.15, 0.47, 0.47])
axins.set_yticklabels([])
axins.set_xticklabels([])
axins.scatter(poincare_position_x_b, poincare_velocity_x_b, color='cyan', s=1)

```

```python

ax.indicate_inset_zoom(axins, edgecolor="black")
mark_inset(ax, axins, loc1=2, loc2=3, fc='none',ec='0.1')

plt.title('Poincare surfaces for stars')
plt.legend()
plt.xlabel('x')
plt.ylabel('$\dot{x}$')
plt.show()

```

```
<>:18: SyntaxWarning: invalid escape sequence '\d'
<>:18: SyntaxWarning: invalid escape sequence '\d'
/var/folders/_3/b44zg9zx3mjcpbfyw4_4d15c0000gn/T/ipykernel_2837/3267452816.py:
  plt.ylabel('$\dot{x}$')
```

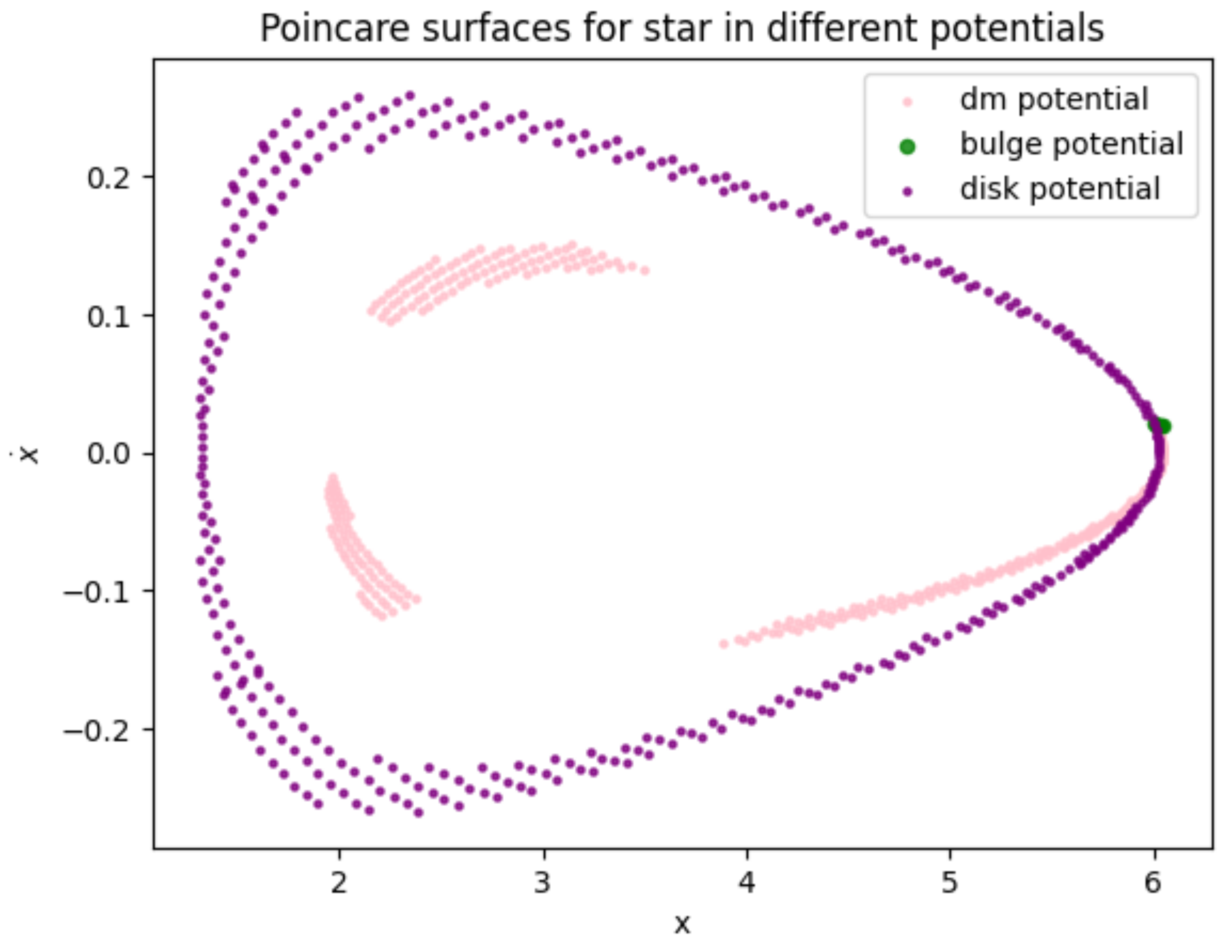



```python
# Disk component
position_x_disk = a_orbit_disk.x.value
velocity_x_disk = a_orbit_disk.vel.d_x.value
position_y_disk = a_orbit_disk.y.value
velocity_y_disk = a_orbit_disk.vel.d_y.value
position_z_disk = a_orbit_disk.z.value
velocity_z_disk = a_orbit_disk.vel.d_z.value

```

```
# Bulge component
position_x_bulge = a_orbit_bulge.x.value
velocity_x_bulge = a_orbit_bulge.vel.d_x.value
position_y_bulge = a_orbit_bulge.y.value
velocity_y_bulge = a_orbit_bulge.vel.d_y.value
position_z_bulge = a_orbit_bulge.z.value
velocity_z_bulge = a_orbit_bulge.vel.d_z.value

# DM component
position_x_dm = a_orbit_DM.x.value
velocity_x_dm = a_orbit_DM.vel.d_x.value
position_y_dm = a_orbit_DM.y.value
velocity_y_dm = a_orbit_DM.vel.d_y.value
position_z_dm = a_orbit_DM.z.value
velocity_z_dm = a_orbit_DM.vel.d_z.value

delta = 0.2

# Selecting particles
poincare_index_disk = np.where((position_y_disk >= -delta) & (position_y_disk
poincare_index_bulge = np.where((position_y_bulge >= -delta) & (position_y_bul
poincare_index_dm = np.where((position_y_dm >= -delta) & (position_y_dm <= del

# Selecting poincare section data for each component
poincare_position_x_disk = position_x_disk[poincare_index_disk]
poincare_velocity_x_disk = velocity_x_disk[poincare_index_disk]
poincare_velocity_y_disk = velocity_y_disk[poincare_index_disk]
poincare_position_y_disk = position_y_disk[poincare_index_disk]
poincare_position_z_disk = position_z_disk[poincare_index_disk]
poincare_velocity_z_disk = velocity_z_disk[poincare_index_disk]

poincare_position_x_bulge = position_x_bulge[poincare_index_bulge]
poincare_velocity_x_bulge = velocity_x_bulge[poincare_index_bulge]
poincare_velocity_y_bulge = velocity_y_bulge[poincare_index_bulge]
poincare_position_y_bulge = position_y_bulge[poincare_index_bulge]
poincare_position_z_bulge = position_z_bulge[poincare_index_bulge]
poincare_velocity_z_bulge = velocity_z_bulge[poincare_index_bulge]

poincare_position_x_dm = position_x_dm[poincare_index_dm]
poincare_velocity_x_dm = velocity_x_dm[poincare_index_dm]
poincare_velocity_y_dm = velocity_y_dm[poincare_index_dm]
poincare_position_y_dm = position_y_dm[poincare_index_dm]
poincare_position_z_dm = position_z_dm[poincare_index_dm]
poincare_velocity_z_dm = velocity_z_dm[poincare_index_dm]
```

```python
fig, ax  = plt.subplots()

ax.scatter(poincare_position_x_dm, poincare_velocity_x_dm,color='pink', alpha=
ax.scatter(poincare_position_x_bulge, poincare_velocity_x_bulge, color='green'
ax.scatter(poincare_position_x_disk, poincare_velocity_x_disk, color='purple',


plt.title('Poincare surfaces for star in different potentials')
plt.legend()
plt.xlabel('x')
plt.ylabel('$\dot{x}$')
```

```
<>:11: SyntaxWarning: invalid escape sequence '\d'
<>:11: SyntaxWarning: invalid escape sequence '\d'
/var/folders/_3/b44zg9zx3mjcpbfyw4_4d15c0000gn/T/ipykernel_2837/3715340710.py:
  plt.ylabel('$\dot{x}$')
Text(0, 0.5, '$\\dot{x}$')
```



3. Utilizar orbita circular en un potencial que contenga solo DM halo y disco.

   $x_i = (x, 0, 0)$

   $v_i = (0, v_{circ}, 0)$

```python
import gala.dynamics as gd
```

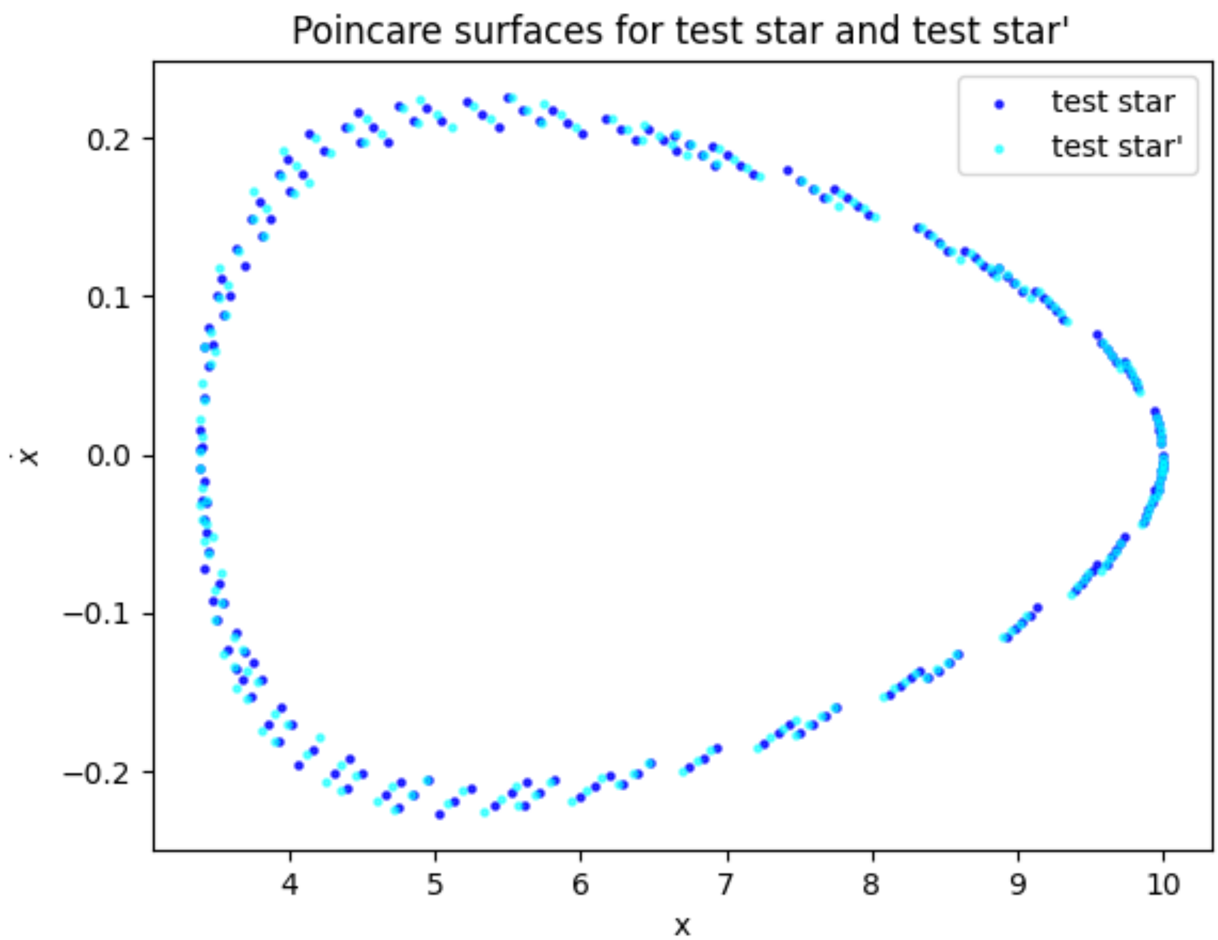
```python
xi = (10,0,0)
vi =  (0,150,0)
ics = gd.PhaseSpacePosition(pos = xi*u.kpc,
                            vel = vi*u.km/u.s)
orbiti = gp.Hamiltonian(total_potential['dm_halo']+total_potential['disk']).in

position_x_i = orbiti.x.value
velocity_x_i = orbiti.vel.d_x.value
position_y_i = orbiti.y.value
velocity_y_i = orbiti.vel.d_y.value
position_z_i = orbiti.z.value
velocity_z_i = orbiti.vel.d_z.value
delta = 0.2

poincare_index_i = np.where((position_y_i >= -delta) & (position_y_i <= delta)
#poincare_index_c = np.where((position_y_c == 0) & (velocity_y_c > 0))[0]

poincare_position_x_i = position_x_i[poincare_index_i]
poincare_velocity_x_i = velocity_x_i[poincare_index_i]
poincare_velocity_y_i = velocity_y_i[poincare_index_i]
poincare_position_y_i = position_y_i[poincare_index_i]
poincare_position_z_i = position_z_i[poincare_index_i]
poincare_velocity_z_i = velocity_z_i[poincare_index_i]
```

4. Perturbar levemente la orbita con un $\Delta v_x << v_y$

```
xii = (10,0,0)
vii =  (-2,150,0)
ics = gd.PhaseSpacePosition(pos = xii*u.kpc,
                            vel = vii*u.km/u.s)
orbitii = gp.Hamiltonian(total_potential['dm_halo']+total_potential['disk']).i

position_x_ii = orbitii.x.value
velocity_x_ii = orbitii.vel.d_x.value
position_y_ii = orbitii.y.value
velocity_y_ii = orbitii.vel.d_y.value
position_z_ii = orbitii.z.value
velocity_z_ii = orbitii.vel.d_z.value
delta = 0.2

poincare_iindex_ii = np.where((position_y_ii >= -delta) & (position_y_ii <= de
#poincare_iindex_c = np.where((position_y_c == 0) & (velocity_y_c > 0))[0]

poincare_position_x_ii = position_x_ii[poincare_iindex_ii]
poincare_velocity_x_ii = velocity_x_ii[poincare_iindex_ii]
poincare_velocity_y_ii = velocity_y_ii[poincare_iindex_ii]
poincare_position_y_ii = position_y_ii[poincare_iindex_ii]
poincare_position_z_ii = position_z_ii[poincare_iindex_ii]
poincare_velocity_z_ii = velocity_z_ii[poincare_iindex_ii]
```

```python
fig, ax  = plt.subplots()

ax.scatter(poincare_position_x_i, poincare_velocity_x_i,color='blue', alpha=0.
ax.scatter(poincare_position_x_ii, poincare_velocity_x_ii,color='cyan', alpha=


plt.title('Poincare surfaces for test star and test star\'')
plt.legend()
plt.xlabel('x')
plt.ylabel('$\dot{x}$')
```

```
<>:10: SyntaxWarning: invalid escape sequence '\d'
<>:10: SyntaxWarning: invalid escape sequence '\d'
/var/folders/_3/b44zg9zx3mjcpbfyw4_4d15c0000gn/T/ipykernel_2837/4103845929.py:
  plt.ylabel('$\dot{x}$')
Text(0, 0.5, '$\\dot{x}$')
```



- Discutir los resultados
  Se nota que la superficie de poincare que representa la estrella levemente pertubada se mueve respecto a la posicion inicial, esto significa que el tipo de orbita es igual (cerrada) pero, las pertubaciones deforman un poco las superficies de poincare.

Practica III

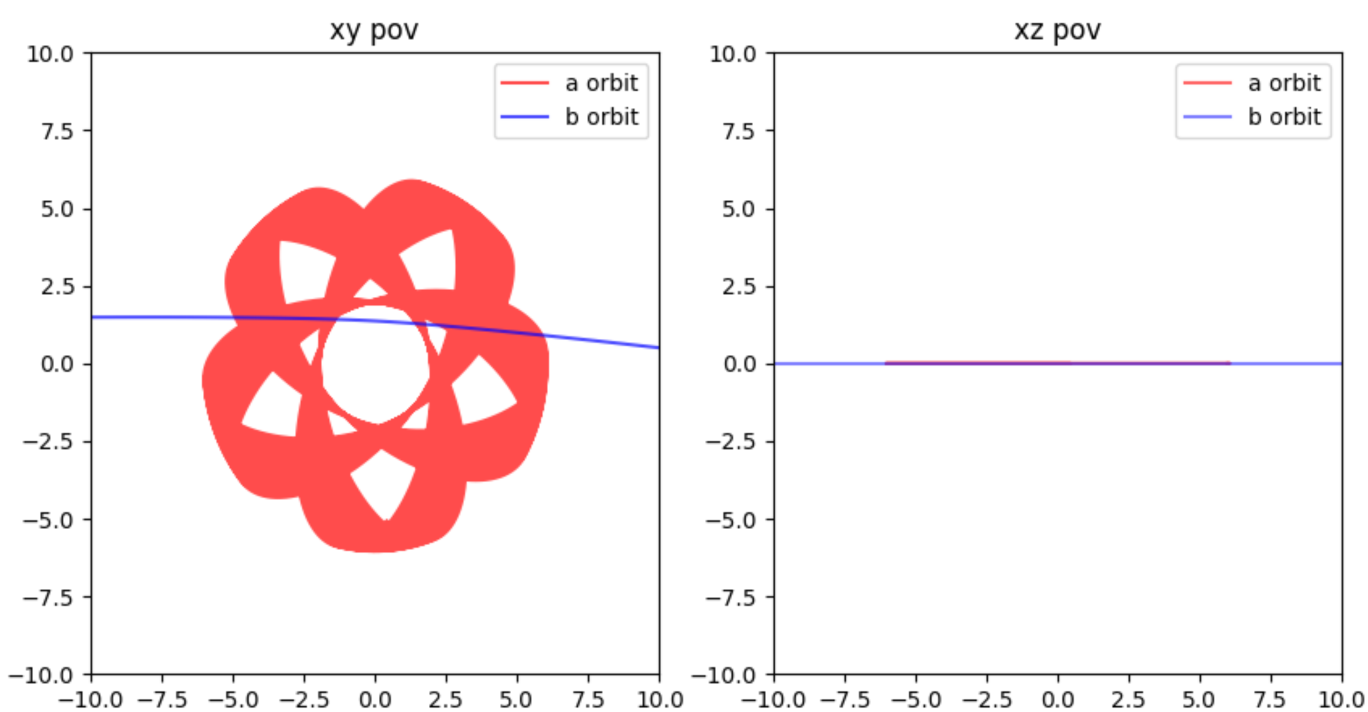
Objetivo: Comprender la densidad localde un enjambre de orbitas y su potencial

1. Utilizando un par de orbitas sobre un potencial esferico de NFW tipo MW, calcular y graficar su evolucion temporal en ~10 Gyrs

Seguiremos utilizando el potencial que tenemos definido que es tipo MW para el DM halo, y las estrellas a (orbita cerrada) y b (orbita hiperbolica)

```python
fig, ax = plt.subplots(1, 2, figsize=(10,5))
ax[0].plot(a_orbit_DM.pos.x ,a_orbit_DM.pos.y,
           color='red', alpha=0.7, label='a orbit')
ax[0].plot(b_orbit_DM.pos.x, b_orbit_DM.pos.y,
           color='blue', alpha=0.7, label='b orbit')
ax[0].set_xlim(-10,10)
ax[0].set_ylim(-10,10)


ax[1].plot(a_orbit_DM.pos.x, a_orbit_DM.pos.z,
           color='red', alpha=0.6, label='a orbit')
ax[1].plot(b_orbit_DM.pos.x, b_orbit_DM.pos.z,
           color='blue', alpha=0.5, label='b orbit')
ax[1].set_xlim(-10,10)
ax[1].set_ylim(-10, 10)

ax[0].set_title('xy pov')
ax[1].set_title('xz pov')

ax[0].legend()
ax[1].legend()
```

```
<matplotlib.legend.Legend at 0x14f53d250>
```

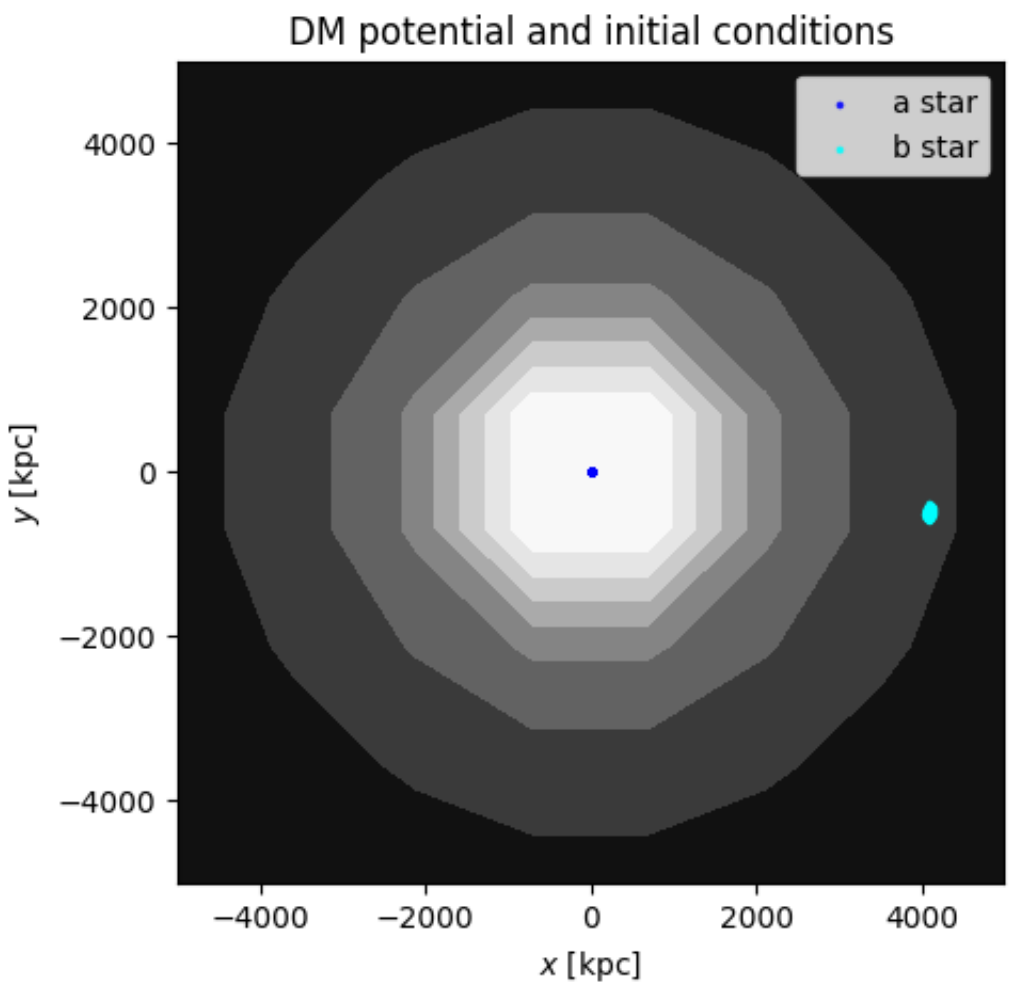


2. Definir un volumen alrededor con las siguientes caracteristicas:
    - Gaussiane en 3D para posiciones y velocidades
    - Condiciones iniciales como la media de posicion y velocidad
    - σ_r y $\sigma_{\dot{r}}$ con orden de 10^{-5} kpc y 1 km/s, respectivamente.
    - Tener en total 1000 condiciones iniciales un poco distintas
    - Integrar todo y observar el enjambre en ~10 Gyrs

```
n_orbits = 1000

ics_a = gd.PhaseSpacePosition(pos = a_star_p*u.kpc,
                              vel = a_star_v*u.km/u.s)
ics_b = gd.PhaseSpacePosition(pos = b_star_p*u.kpc,
                              vel = b_star_v*u.km/u.s)


new_pos_a = np.random.normal(ics_a.pos.xyz.to(u.pc).value, 0.01,
                             size=(n_orbits,3)).T*u.pc
new_vel_a = np.random.normal(ics_a.vel.d_xyz.to(u.km/u.s).value, 1,
                             size=(n_orbits,3)).T * u.km/u.s

new_ics_a = gd.PhaseSpacePosition(pos=new_pos_a, vel=new_vel_a)
orbit_a = gp.Hamiltonian(total_potential['dm_halo']).integrate_orbit(ics_a,  d

orbits_total_a = gp.Hamiltonian(total_potential['dm_halo']).integrate_orbit(ne


new_pos_b = np.random.normal(ics_b.pos.xyz.to(u.pc).value, 0.01,
                               size=(n_orbits,3)).T*u.pc
new_vel_b = np.random.normal(ics_b.vel.d_xyz.to(u.km/u.s).value, 1.,
                             size=(n_orbits,3)).T * u.km/u.s

new_ics_b = gd.PhaseSpacePosition(pos=new_pos_b, vel=new_vel_b)
orbit_b = gp.Hamiltonian(total_potential['dm_halo']).integrate_orbit(ics_b,  d

orbits_total_b = gp.Hamiltonian(total_potential['dm_halo']).integrate_orbit(ne
```

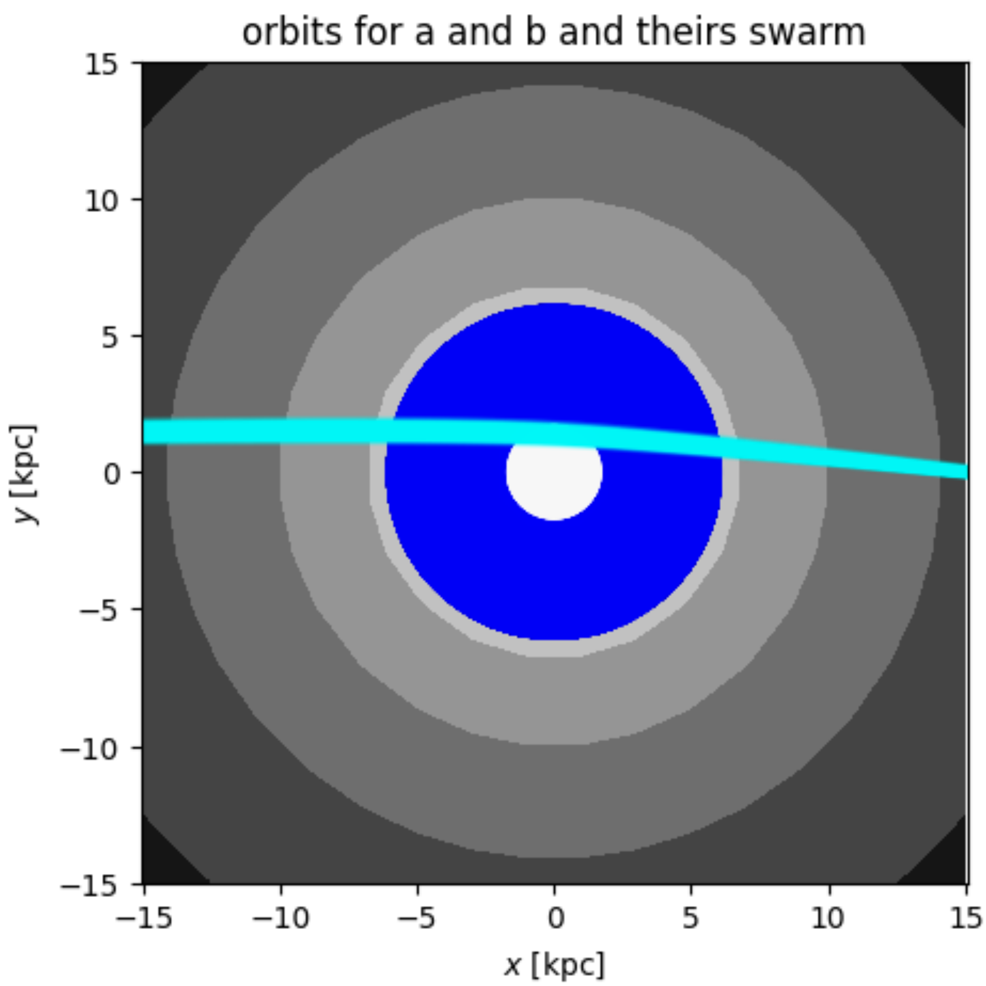
```
grid = np.linspace(-5000,5000,8)
fig, ax = plt.subplots(1, 1, figsize=(5,5))
fig = total_potential['dm_halo'].plot_contours(grid=(grid, grid, 0), cmap='Gre
fig = orbits_total_a[-1].plot(['x','y'], color='blue', s=10,
                          alpha=0.8, axes=[ax], auto_aspect=False,
                          label='a star')
fig = orbits_total_b[-1].plot(['x','y'], color='cyan', s=10,
                                   alpha=0.8, axes=[ax], auto_aspect=False,
                                   label='b star')
ax.set_xlim(-5000,5000)
ax.set_ylim(-5000,5000)
ax.legend()
ax.set_title('DM potential and initial conditions')
```

```
Text(0.5, 1.0, 'DM potential and initial conditions')
```



```
grid = np.linspace(-15,15,16)
fig, ax = plt.subplots(1, 1, figsize=(5,5))
fig = total_potential['dm_halo'].plot_contours(grid=(grid, grid, 0), cmap='Gre
fig = orbits_total_a.plot(['x','y'], color='blue', axes=[ax], alpha=0.1, label
fig = orbits_total_b.plot(['x','y'], color='cyan', axes=[ax], alpha=0.1, label
ax.set_title('orbits for a and b and theirs swarm')
ax.set_xlim(-15, 15)
ax.set_ylim(-15, 15)
```

```
(-15.0, 15.0)
```



3. Graficar y observar evolucion de σ_r y $\sigma_{\dot{r}}$.

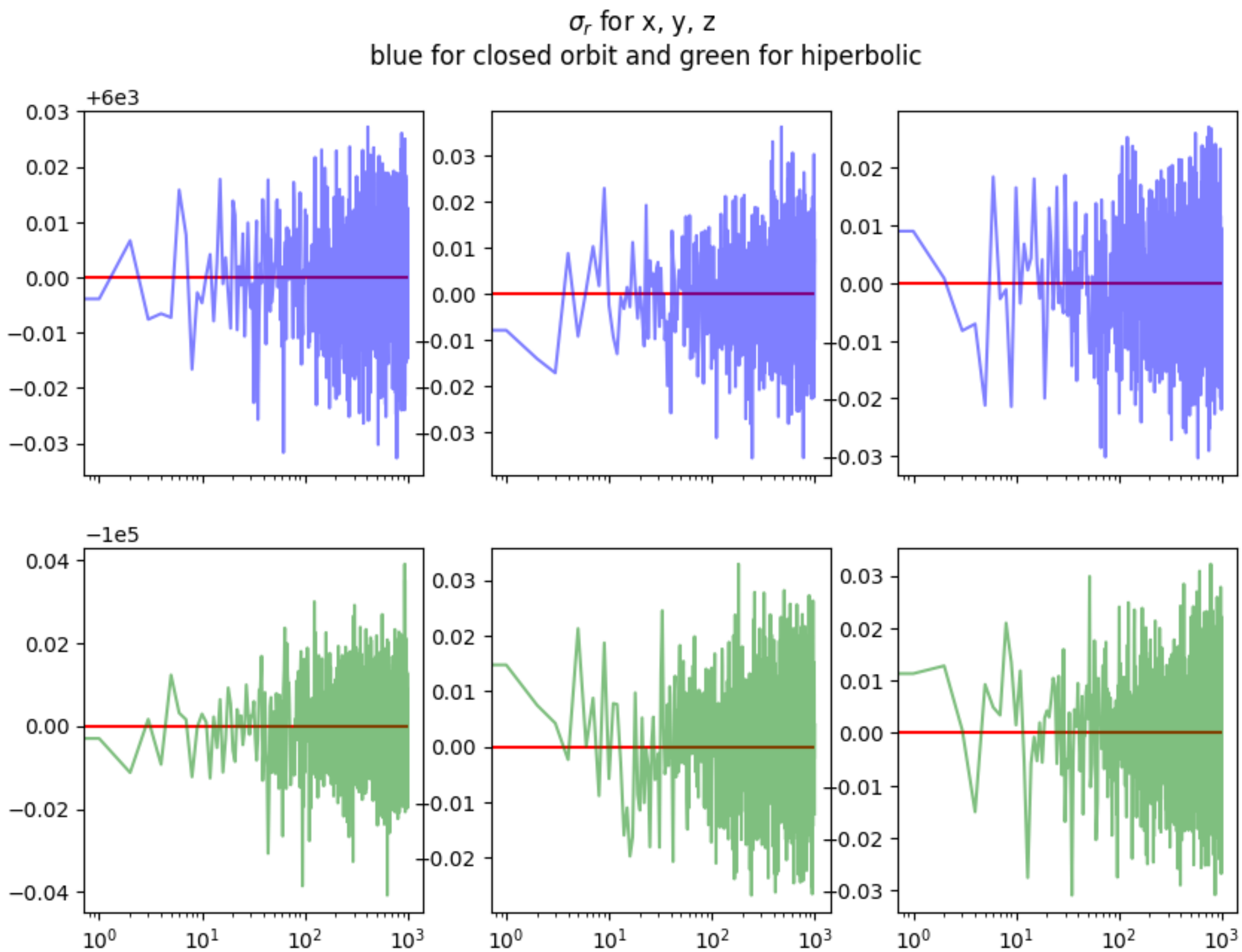
- sigma total
- sigma por componente ([x,y,z], [$\dot{x}$, $\dot{y}$, $\dot{z}$])

Nuestros parametros iniciales son:

- a_star_p = [6,0,0]
- a_star_v = [20,80,0]
- b_star_p = [-100,0,0]
- b_star_v = [500,10,0]
  Las cuales estan en kiloparsec. Los σ seran graficados en parsec. Por ende los errores son muy pequenitos.

```
fig, ax = plt.subplots(2,3, sharex=True, figsize=(10,7))
xdata= np.arange(0,1000)

#x
ax[0,0].hlines(y=6000, xmin=0, xmax=1000, color='red')
ax[0,0].plot(xdata, new_pos_a[0], color='blue', alpha=0.5)

ax[0,0].set_xscale('log')
ax[1,0].hlines(y=-100000, xmin=0, xmax=1000, color='red')
ax[1,0].plot(xdata, new_pos_b[0], color='green', alpha=0.5)

#y
ax[0,1].hlines(y=0, xmin=0, xmax=1000, color='red')
ax[0,1].plot(xdata, new_pos_a[1], color='blue', alpha=0.5)

ax[1,1].hlines(y=0, xmin=0, xmax=1000, color='red')
ax[1,1].plot(xdata, new_pos_b[1], color='green', alpha=0.5)


#z
ax[0,2].hlines(y=0, xmin=0, xmax=1000, color='red')
ax[0,2].plot(xdata, new_pos_a[2], color='blue', alpha=0.5)

ax[1,2].hlines(y=0, xmin=0, xmax=1000, color='red')
ax[1,2].plot(xdata, new_pos_b[2], color='green', alpha=0.5)

fig.suptitle('$\sigma_{r}$ for x, y, z \n blue for closed orbit and green for
```

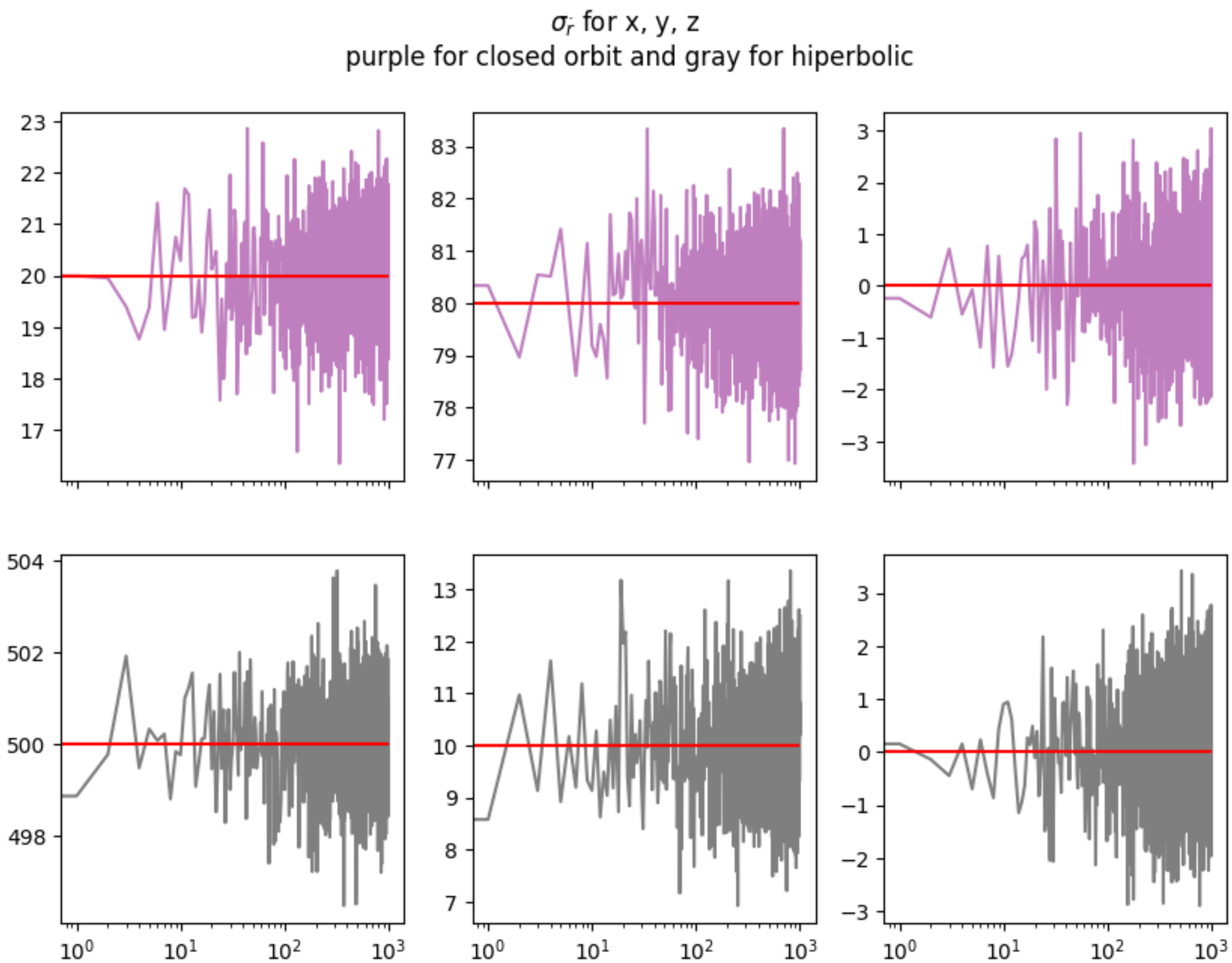
```
<>:27: SyntaxWarning: invalid escape sequence '\s'
<>:27: SyntaxWarning: invalid escape sequence '\s'
/var/folders/_3/b44zg9zx3mjcpbfyw4_4d15c0000gn/T/ipykernel_2837/521037230.py:2
  fig.suptitle('$\sigma_{r}$ for x, y, z \n blue for closed orbit and green fo
Text(0.5, 0.98, '$\\sigma_{r}$ for x, y, z \n blue for closed orbit and green
for hiperbolic')
```




```
fig, ax = plt.subplots(2,3, sharex=True, figsize=(10,7))
xdata= np.arange(0,1000)

ax[0,0].set_xscale('log')

ax[0,0].plot(xdata, new_vel_a[0], color='purple', alpha=0.5)
ax[0,0].hlines(y=20, xmin=0, xmax=1000, color='red')
```

```
ax[1,0].plot(xdata, new_vel_b[0], color='black', alpha=0.5)
ax[1,0].hlines(y=500,xmin=0, xmax=1000, color='red')

ax[0,1].plot(xdata, new_vel_a[1], color='purple', alpha=0.5)
ax[0,1].hlines(y=80, xmin=0, xmax=1000, color='red')

ax[1,1].plot(xdata, new_vel_b[1], color='black', alpha=0.5)
ax[1,1].hlines(y=10, xmin=0, xmax=1000, color='red')

ax[0,2].plot(xdata, new_vel_a[2], color='purple', alpha=0.5)
ax[0,2].hlines(y=0, xmin=0, xmax=1000, color='red')

#0
ax[1,2].plot(xdata, new_vel_b[2], color='black', alpha=0.5)
ax[1,2].hlines(y=0, xmin=0, xmax=1000, color='red')

fig.suptitle('$\sigma_{\dot{r}}$ for x, y, z \n purple for closed orbit and gr
```

```
<>:25: SyntaxWarning: invalid escape sequence '\s'
<>:25: SyntaxWarning: invalid escape sequence '\s'
/var/folders/_3/b44zg9zx3mjcpbfyw4_4d15c0000gn/T/ipykernel_2837/1194764760.py:
  fig.suptitle('$\sigma_{\dot{r}}$ for x, y, z \n purple for closed orbit and
Text(0.5, 0.98, '$\\sigma_{\\dot{r}}$ for x, y, z \n purple for closed orbit
and gray for hiperbolic')
```



Si observamos y superponemos los graficos de posicion y velocidad para la misma componente donde ambos parten en cero, se puede notar lo visto en clases del paper de Gomez+2013, es decir, el comportamiento de x y $\dot{x}$ como sin y cos (ya que $\dot{x}$ es la derivada de x)

4. Tomar esferas de 1 y 2,5 kpc de radio, centradas a todo tiempo t, en la órbita central del enjambre.
   En cada paso de integración, calcular la densidad de partículas dentro de estas esferas.

```
fig, anim = orbits_total_a[:1000].cylindrical.animate(components=['rho', 'z'],
                                                     stride=10)
anim.save('closedswarmorbit.gif')
plt.close()
```

```
MovieWriter ffmpeg unavailable; using Pillow instead.
```

```
fig, anim = orbits_total_b[:1000].cylindrical.animate(components=['rho', 'z'],
                                                     stride=10)
anim.save('hiperbolicswarmorbit.gif')
plt.close()
```

```
MovieWriter ffmpeg unavailable; using Pillow instead.
```

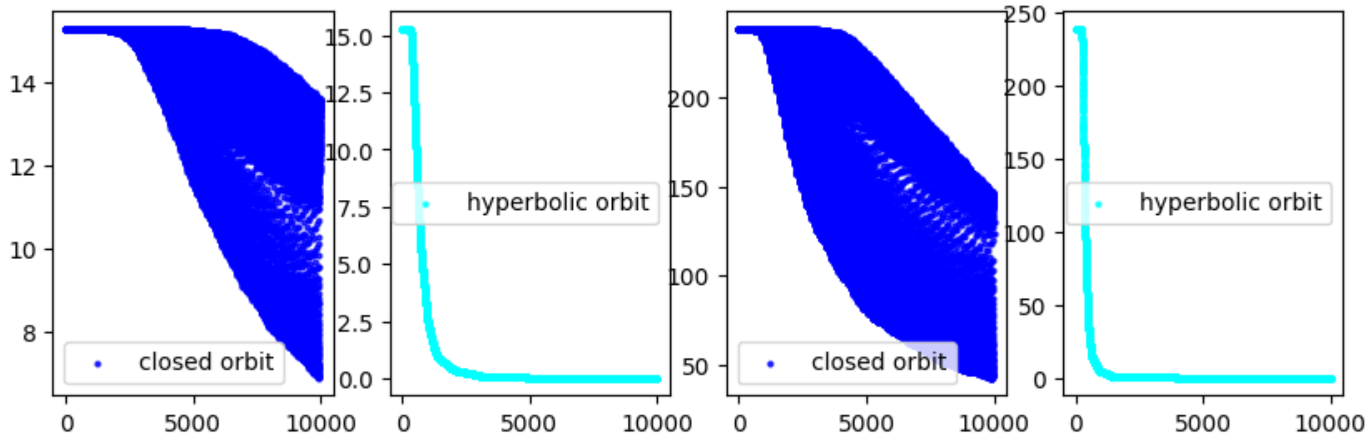
Con las animaciones credas en la celda anterior, (closedswamorbit.gif y hiperbolicswarmorbit.gif), podemos observar que la densidad del enjambre aumenta a medida que se acerca mas a la parte mas masiva del potencial, y disminuye a medida que se aleja.

```
density25 = []
density10 = []

for i in range(len(orbits_total_a.t)):
    difference = orbits_total_a.pos[i,:] - orbit_a.pos[i]
    distance_25 = difference.norm()/(2.5*u.kpc)
    distance_10 = difference.norm()/(1.0*u.kpc)

    pos_25 = np.where(distance_25<=1)[0]
    pos_10 = np.where(distance_10<=1)[0]

    densitypoints_25 = distance_25[pos_25]
    densitypoints_10 = distance_10[pos_10]

    vol25 = (4/3) * np.pi * (2.5**3)
    densitytotal_25 = len(densitypoints_25) / vol25
    density25.append(densitytotal_25)

    vol10 = (4/3) * np.pi * (1.0**3)
    densitytotal_10 = len(densitypoints_10) / vol10
    density10.append(densitytotal_10)
```

```
density25b = []
density10b = []

for i in range(len(orbits_total_b.t)):
    difference = orbits_total_b.pos[i,:] - orbit_b.pos[i]
    distance_25b = difference.norm()/(2.5*u.kpc)
    distance_10b = difference.norm()/(1.0*u.kpc)

    pos_25b = np.where(distance_25b<=1)[0]
    pos_10b = np.where(distance_10b<=1)[0]

    densitypoints_25b = distance_25b[pos_25b]
    densitypoints_10b = distance_10b[pos_10b]

    vol25b = (4/3) * np.pi * (2.5**3)
    densitytotal_25b = len(densitypoints_25b) / vol25b
    density25b.append(densitytotal_25b)

    vol10b = (4/3) * np.pi * (1.0**3)
    densitytotal_10b = len(densitypoints_10b) / vol10b
    density10b.append(densitytotal_10b)
```

```
fig, ax = plt.subplots(1,4, figsize=(10,3))
ax[0].scatter(orbits_total_a.t, density25, s=4,
              alpha=0.8, color='blue', label='closed orbit')
ax[1].scatter(orbits_total_b.t, density25b, s=4,
              alpha=0.8, color='cyan', label='hyperbolic orbit' )

ax[2].scatter(orbits_total_a.t, density10,
              s=4, alpha=0.8, color='blue', label='closed orbit')
ax[3].scatter(orbits_total_b.t, density10b,
              s=4, alpha=0.8, color='cyan', label='hyperbolic orbit')
ax[0].legend()
ax[1].legend()
ax[2].legend()
ax[3].legend()
```

```
<matplotlib.legend.Legend at 0x30309d6a0>
```



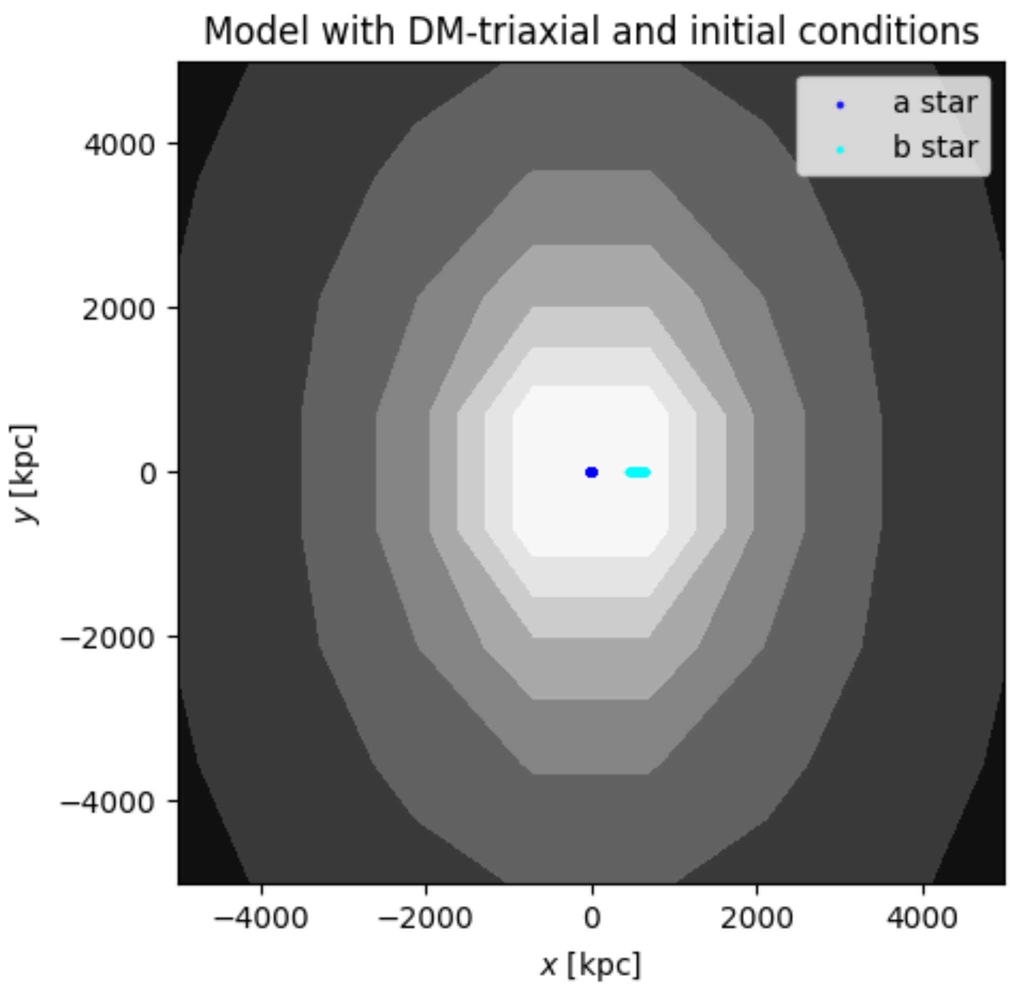
5. Repetir el experimento, ahora considerando:
    - Un potencial NFW triaxial.
    - Un potencial MW-like (compueso completo)

```
total_potential_triaxial = gp.CCompositePotential()
total_potential_triaxial['disk'] = gp.MiyamotoNagaiPotential(m = 1E11 , a=3, b
total_potential_triaxial['bulge'] = gp.HernquistPotential(m = 3E9 , c = 0.67,
total_potential_triaxial = gp.NFWPotential(m=5E12,r_s=80, a=2, b=3, c=7, units
```

```
n_orbits = 1000


ics_a = a_particle
ics_b = b_particle


new_pos_a = np.random.normal(ics_a.pos.xyz.to(u.pc).value, 0.01,
                           size=(n_orbits,3)).T*u.pc
new_vel_a = np.random.normal(ics_a.vel.d_xyz.to(u.km/u.s).value, 1,
                           size=(n_orbits,3)).T * u.km/u.s

new_ics_a = gd.PhaseSpacePosition(pos=new_pos_a, vel=new_vel_a)
orbit_a = gp.Hamiltonian(total_potential_triaxial).integrate_orbit(ics_a,  dt=

orbits_total_a = gp.Hamiltonian(total_potential_triaxial).integrate_orbit(new_


new_pos_b = np.random.normal(ics_b.pos.xyz.to(u.pc).value, 0.01,
                             size=(n_orbits,3)).T*u.pc
new_vel_b = np.random.normal(ics_b.vel.d_xyz.to(u.km/u.s).value, 1.,
                           size=(n_orbits,3)).T * u.km/u.s

new_ics_b = gd.PhaseSpacePosition(pos=new_pos_b, vel=new_vel_b)
orbit_b = gp.Hamiltonian(total_potential_triaxial).integrate_orbit(ics_b,  dt=

orbits_total_b = gp.Hamiltonian(total_potential_triaxial).integrate_orbit(new_
```

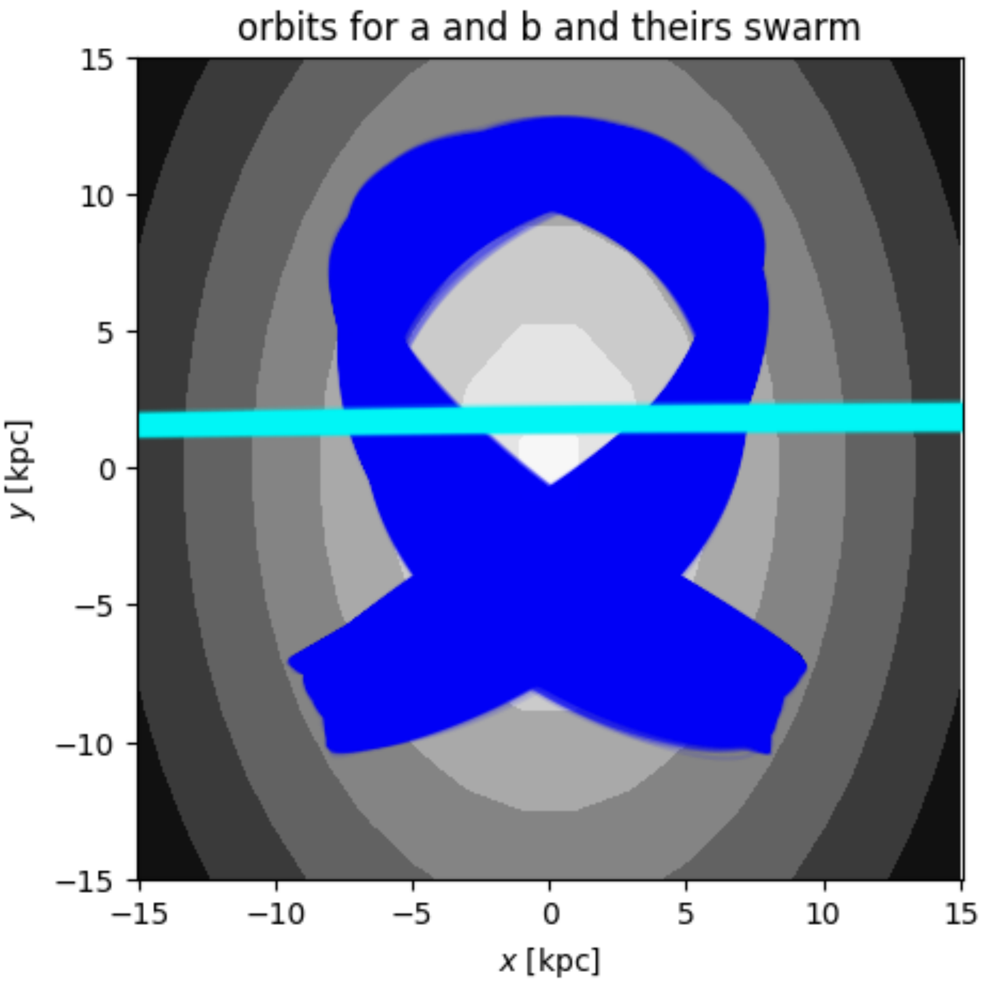
```
grid = np.linspace(-5000,5000,8)
fig, ax = plt.subplots(1, 1, figsize=(5,5))
fig = total_potential_triaxial.plot_contours(grid=(grid, grid, 0), cmap='Greys
fig = orbits_total_a[-1].plot(['x','y'], color='blue', s=10,
                        alpha=0.8, axes=[ax], auto_aspect=False,
                        label='a star')
fig = orbits_total_b[-1].plot(['x','y'], color='cyan', s=10,
                                alpha=0.8, axes=[ax], auto_aspect=False,
                                label='b star')
ax.set_xlim(-5000,5000)
ax.set_ylim(-5000,5000)
ax.legend()
ax.set_title('Model with DM-triaxial and initial conditions')
```

```
Text(0.5, 1.0, 'Model with DM-triaxial and initial conditions')
```



```
grid = np.linspace(-15,15,16)
fig, ax = plt.subplots(1, 1, figsize=(5,5))
fig = total_potential_triaxial.plot_contours(grid=(grid, grid, 0), cmap='Greys
fig = orbits_total_a.plot(['x','y'], color='blue', axes=[ax], alpha=0.1, label
fig = orbits_total_b.plot(['x','y'], color='cyan', axes=[ax], alpha=0.1, label
ax.set_title('orbits for a and b and theirs swarm')
ax.set_xlim(-15, 15)
ax.set_ylim(-15, 15)
```

```
(-15.0, 15.0)
```



3. Graficar y observar evolucion de σ_r y $\sigma_{\dot{r}}$.

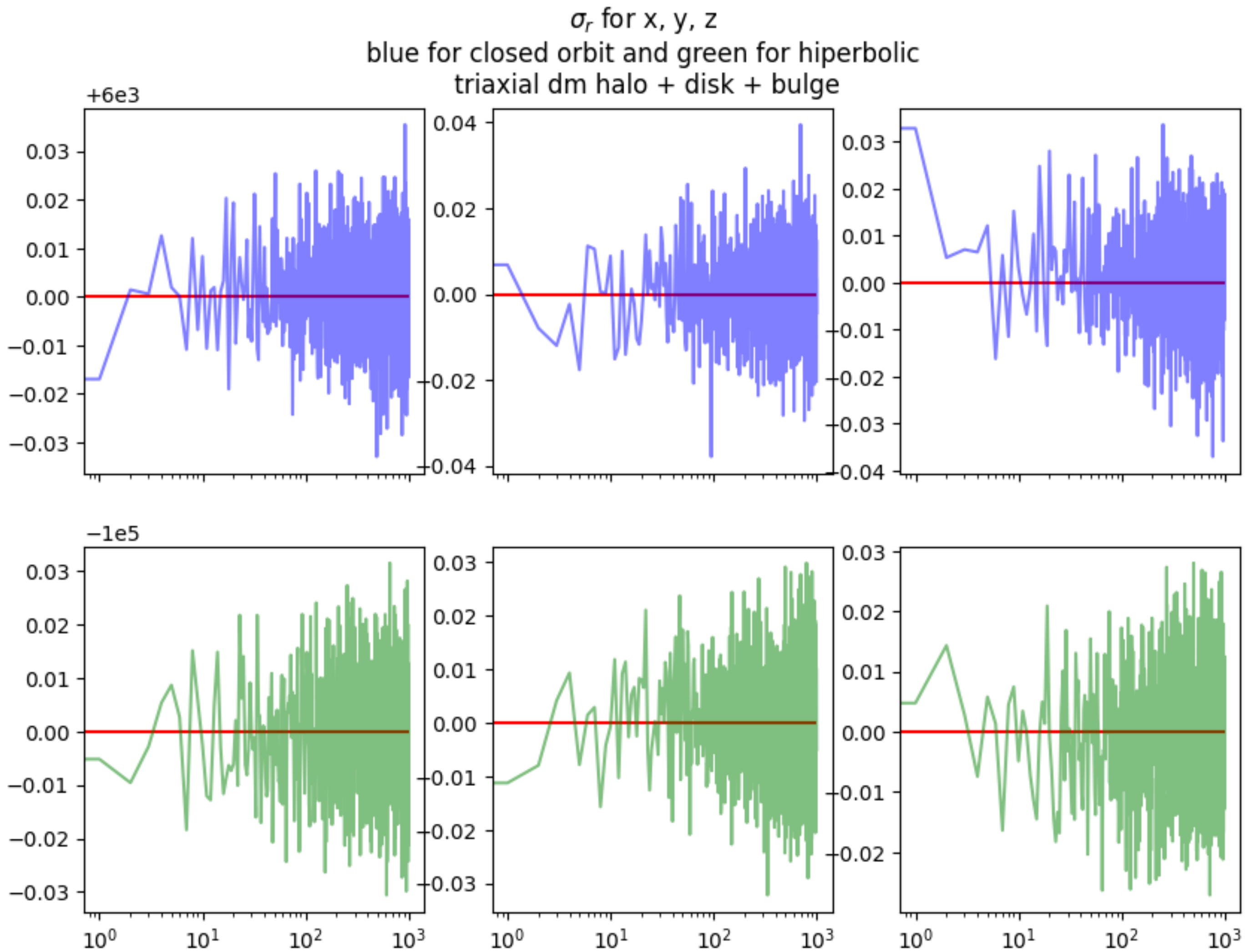
- sigma total
- sigma por componente ([x,y,z], [$\dot{x}$, $\dot{y}$, $\dot{z}$])

Nuestros parametros iniciales son:

- a_star_p = [6,0,0]
- a_star_v = [20,80,0]
- b_star_p = [-100,0,0]
- b_star_v = [500,10,0]
  Las cuales estan en kiloparsec. Los σ seran graficados en parsec. Por ende los errores son muy pequenitos.

```
fig, ax = plt.subplots(2,3, sharex=True, figsize=(10,7))
xdata= np.arange(0,1000)

#x
ax[0,0].hlines(y=6000, xmin=0, xmax=1000, color='red')
ax[0,0].plot(xdata, new_pos_a[0], color='blue', alpha=0.5)

ax[0,0].set_xscale('log')
ax[1,0].hlines(y=-100000, xmin=0, xmax=1000, color='red')
ax[1,0].plot(xdata, new_pos_b[0], color='green', alpha=0.5)

#y
ax[0,1].hlines(y=0, xmin=0, xmax=1000, color='red')
ax[0,1].plot(xdata, new_pos_a[1], color='blue', alpha=0.5)

ax[1,1].hlines(y=0, xmin=0, xmax=1000, color='red')
ax[1,1].plot(xdata, new_pos_b[1], color='green', alpha=0.5)


#z
ax[0,2].hlines(y=0, xmin=0, xmax=1000, color='red')
ax[0,2].plot(xdata, new_pos_a[2], color='blue', alpha=0.5)

ax[1,2].hlines(y=0, xmin=0, xmax=1000, color='red')
ax[1,2].plot(xdata, new_pos_b[2], color='green', alpha=0.5)

fig.suptitle('$\sigma_{r}$ for x, y, z \n blue for closed orbit and green for




```

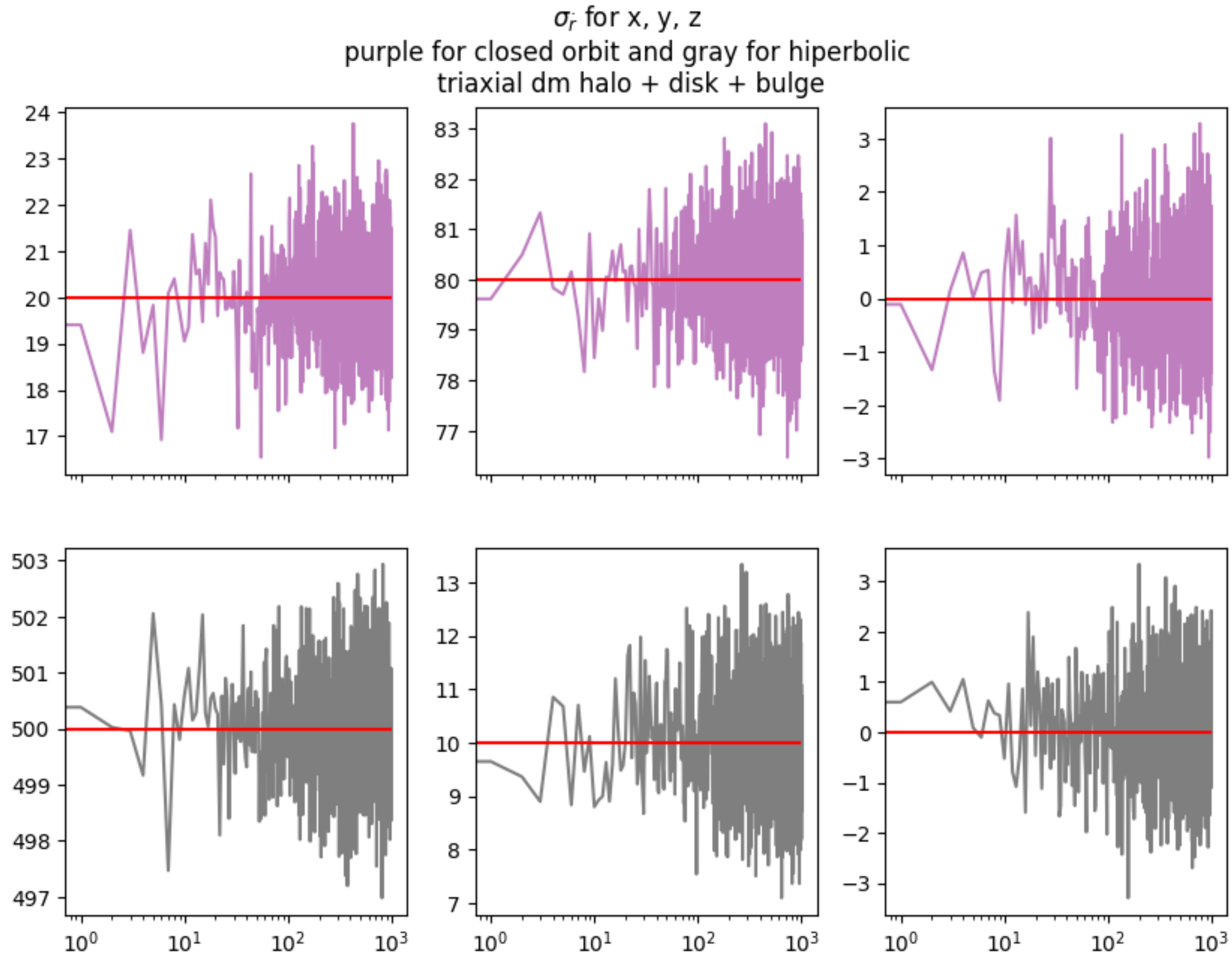
```
<>:27: SyntaxWarning: invalid escape sequence '\s'
<>:27: SyntaxWarning: invalid escape sequence '\s'
/var/folders/_3/b44zg9zx3mjcpbfyw4_4d15c0000gn/T/ipykernel_2837/4119965136.py:
  fig.suptitle('$\sigma_{r}$ for x, y, z \n blue for closed orbit and green fc
Text(0.5, 0.98, '$\\sigma_{r}$ for x, y, z \n blue for closed orbit and green
for hiperbolic \n triaxial dm halo + disk + bulge')
```




```
fig, ax = plt.subplots(2,3, sharex=True, figsize=(10,7))
xdata= np.arange(0,1000)

ax[0,0].set_xscale('log')

ax[0,0].plot(xdata, new_vel_a[0], color='purple', alpha=0.5)
ax[0,0].hlines(y=20, xmin=0, xmax=1000, color='red')
```

```
ax[1,0].plot(xdata, new_vel_b[0], color='black', alpha=0.5)
ax[1,0].hlines(y=500,xmin=0, xmax=1000, color='red')

ax[0,1].plot(xdata, new_vel_a[1], color='purple', alpha=0.5)
ax[0,1].hlines(y=80, xmin=0, xmax=1000, color='red')

ax[1,1].plot(xdata, new_vel_b[1], color='black', alpha=0.5)
ax[1,1].hlines(y=10, xmin=0, xmax=1000, color='red')

ax[0,2].plot(xdata, new_vel_a[2], color='purple', alpha=0.5)
ax[0,2].hlines(y=0, xmin=0, xmax=1000, color='red')

#0
ax[1,2].plot(xdata, new_vel_b[2], color='black', alpha=0.5)
ax[1,2].hlines(y=0, xmin=0, xmax=1000, color='red')

fig.suptitle('$\sigma_{\dot{r}}$ for x, y, z \n purple for closed orbit and gr
```

```
<>:25: SyntaxWarning: invalid escape sequence '\s'
<>:25: SyntaxWarning: invalid escape sequence '\s'
/var/folders/_3/b44zg9zx3mjcpbfyw4_4d15c0000gn/T/ipykernel_2837/2812287091.py:
  fig.suptitle('$\sigma_{\dot{r}}$ for x, y, z \n purple for closed orbit and
Text(0.5, 0.98, '$\\sigma_{\\dot{r}}$ for x, y, z \n purple for closed orbit
and gray for hiperbolic \n triaxial dm halo + disk + bulge')
```



Si observamos y superponemos los graficos de posicion y velocidad para la misma componente donde ambos parten en cero, se puede notar lo visto en clases del paper de Gomez+2013, es decir, el comportamiento de x y $\dot{x}$ como sin y cos (ya que $\dot{x}$ es la derivada de x)

4. Tomar esferas de 1 y 2,5 kpc de radio, centradas a todo tiempo t, en la órbita central del enjambre.
   En cada paso de integración, calcular la densidad de partículas dentro de estas esferas.

```
fig, anim = orbits_total_a[:1000].cylindrical.animate(components=['rho', 'z'],
                                                      stride=10)
anim.save('closedswarmorbit_triaxial.gif')
plt.close()
```

```
MovieWriter ffmpeg unavailable; using Pillow instead.
```

```
fig, anim = orbits_total_b[:1000].cylindrical.animate(components=['rho', 'z'],
                                                      stride=10)
anim.save('hiperbolicswarmorbit_triaxial.gif')
plt.close()
```

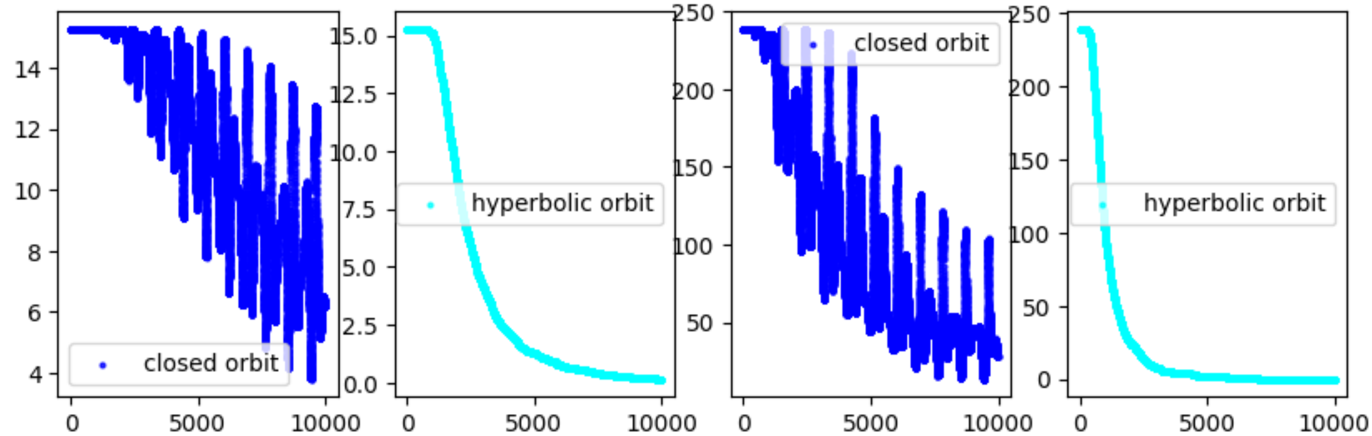
```
MovieWriter ffmpeg unavailable; using Pillow instead.
```

```
density25 = []
density10 = []

for i in range(len(orbits_total_a.t)):
    difference = orbits_total_a.pos[i,:] - orbit_a.pos[i]
    distance_25 = difference.norm()/(2.5*u.kpc)
    distance_10 = difference.norm()/(1.0*u.kpc)

    pos_25 = np.where(distance_25<=1)[0]
    pos_10 = np.where(distance_10<=1)[0]

    densitypoints_25 = distance_25[pos_25]
    densitypoints_10 = distance_10[pos_10]

    vol25 = (4/3) * np.pi * (2.5**3)
    densitytotal_25 = len(densitypoints_25) / vol25
    density25.append(densitytotal_25)

    vol10 = (4/3) * np.pi * (1.0**3)
    densitytotal_10 = len(densitypoints_10) / vol10
    density10.append(densitytotal_10)
```

```python
density25b = []
density10b = []

for i in range(len(orbits_total_b.t)):
    difference = orbits_total_b.pos[i,:] - orbit_b.pos[i]
    distance_25b = difference.norm()/(2.5*u.kpc)
    distance_10b = difference.norm()/(1.0*u.kpc)

    pos_25b = np.where(distance_25b<=1)[0]
    pos_10b = np.where(distance_10b<=1)[0]

    densitypoints_25b = distance_25b[pos_25b]
    densitypoints_10b = distance_10b[pos_10b]

    vol25b = (4/3) * np.pi * (2.5**3)
    densitytotal_25b = len(densitypoints_25b) / vol25b
    density25b.append(densitytotal_25b)

    vol10b = (4/3) * np.pi * (1.0**3)
    densitytotal_10b = len(densitypoints_10b) / vol10b
    density10b.append(densitytotal_10b)
```

```python
fig, ax = plt.subplots(1,4, figsize=(10,3))
ax[0].scatter(orbits_total_a.t, density25, s=4,
              alpha=0.8, color='blue', label='closed orbit')
ax[1].scatter(orbits_total_b.t, density25b, s=4,
              alpha=0.8, color='cyan', label='hyperbolic orbit' )

ax[2].scatter(orbits_total_a.t, density10,
              s=4, alpha=0.8, color='blue', label='closed orbit')
ax[3].scatter(orbits_total_b.t, density10b,
              s=4, alpha=0.8, color='cyan', label='hyperbolic orbit')
ax[0].legend()
ax[1].legend()
ax[2].legend()
ax[3].legend()
```

```
<matplotlib.legend.Legend at 0x16444fef0>
```



Finalmente podemos observar que en el caso de que el potencial cambie, las orbitas van a verse afectadas al punto de volverse abiertas como es el caso.

Double-click (or enter) to edit