orbittools

Orbittools is a set of functions useful in working with 2-body problems and observations. It's not not comprehensive nor particularly fancy, but it is useful. Bascially I wanted a place to store and easily call functions I used all the time. I'll update it sometimes.

In [1]:
```
from orbittools.orbittools import *
```

orbittools contains some basic functions that it's nice to automate. Here I'll show what they all do and what inputs look like.

period uses Kepler's third law to compute the period of a test particle with a certain semi-major axis in a Keplerian orbit around a central mass. It can take astropy unit objects of any distance and mass and returns period in years:

In [2]:
```
period(1*u.au,1*u.Msun)
```

Out[2]: 1 yr

In [3]:
```
period(149.60e6*u.km,2e30*u.kg)
```

Out[3]: 0.99713598 yr

Or if you enter values without units, it will return a number in years without an astropy unit. You must enter semi-major axis in au and mass in solar masses to get the right answer

In [4]:
```
period(1,1)
```

Out[4]:
```
1.0
```

In [5]:
```
period(149.60e6,2e30)
```

Out[5]:
```
0.0012938454188967088
```

distance uses the Bayesian estimation formulation given in Bailer-Jones 2015 to compute distance + error in parsecs given parallax + error in mas. Designed to work with the output of Gaia parallaxes.

For example the distance to HR 8799 using Gaia's parallax is:

In [6]:
```
distance(24.217514232723282,0.088094235139766 26)
```

Out[6]:
```
(41.29350566835295, 0.15020740717277492)
```

to_polar converts RA/DEC in degrees of two objects into their relative separation in mas and position angle in degrees.

For example, the wide stellar binary DS Tuc A and B both have well-defined solutions in Gaia DR2, and their separation and position angle is:

In [7]:
```
deg_to_mas = 3600000.
mas_to_deg = 1./3600000.

RAa, RAaerr = 354.9154672903039, 0.03459837195273078*mas_to_deg
DECa, DECaerr = -69.19604296286967, 0.02450688383611924*mas_to_deg
RAb, RAberr = 354.914570528965, 0.028643873627224457*mas_to_deg
DECb, DECberr = -69.19458723113503, 0.01971674184397741*mas_to_deg

to_polar(RAa,RAb,DECa,DECb)
```

Out[7]:
```
(<Quantity 5364.61187229 mas>, <Quantity 347.65815486 deg>)
```

You can do a quick Monte Carlo to get errors:

In [8]:
```
seppa = to_polar(np.random.normal(RAa,RAaerr,10000),
                 np.random.normal(RAb,RAberr,10000),
                 np.random.normal(DECa,DECaerr,10000),
                 np.random.normal(DECb,DECberr,10000))

print('Separation =',np.median(seppa[0]),'+-',np.std(seppa[0]))
print('PA =',np.median(seppa[1]),'+-',np.std(seppa[1]))
```

```
Separation = 5364.612076426886 mas +- 0.030608627492292793 mas
PA = 347.6581557948317 deg +- 0.00017997946638202 36 deg
```

physical_separation takes in the distance and angular separation between two objects and returns their physical separation in au. Distance and angle must be astropy units.

In [9]:
```
physical_separation(4.5*u.lyr,230*u.mas)
```

Out[9]: 0.31733244 AU

angular_separation takes in distance and physical separation and returns angular separation in arcsec. Distance and separation must be in astropy units.

In [10]:
```
angular_separation(4.5*u.lyr,0.317*u.au)
```

Out[10]: 0.22975905 ″

keplerian_to_cartesian takes in keplerian orbital elements and returns the observed 3D position, velocity, and acceleration vectors in a right-handed system with +X = +DEC, +Y = +RA, +Z = towards the observer.

Let's look at the inputs:

In [11]:
```
help(keplerian_to_cartesian)
```

```
Help on function keplerian_to_cartesian in module orbittools.orbittools:

keplerian_to_cartesian(sma, ecc, inc, argp, lon, meananom, kep)
    Given a set of Keplerian orbital elements, returns the observable 3-d
imensional position, velocity,
    and acceleration at the specified time.  Accepts and arbitrary number
of input orbits.  Semi-major
    axis must be an astropy unit object in physical distance (ex: au, but
not arcsec).  The observation
    time must be converted into mean anomaly before passing into functio
n.
    Inputs:
        sma (1xN arr flt) [au]: semi-major axis in au, must be an astropy
units object
        ecc (1xN arr flt) [unitless]: eccentricity
        inc (1xN arr flt) [deg]: inclination
        argp (1xN arr flt) [deg]: argument of periastron
        lon (1xN arr flt) [deg]: longitude of ascending node
        meananom (1xN arr flt) [radians]: mean anomaly
        kep (1xN arr flt): kepler constant = mu/m where mu = G*m1*m2 and
m = [1/m1 + 1/m2]^-1 .
                    In the limit of m1>>m2, mu = G*m1 and m = m2
    Returns:
        pos (3xN arr) [au]: position in xyz coords in au, with
                    x = pos[0], y = pos[1], z = pos[2] for each of N orbi
ts
                    +x = +Dec, +y = +RA, +z = towards observer
        vel (3xN arr) [km/s]: velocity in xyz plane.
        acc (3xN arr) [km/s/yr]: acceleration in xyz plane.
    Written by Logan Pearce, 2019, inspired by Sarah Blunt
```

In [12]:
```
sma = 5.2*u.au
ecc = 0.2
inc = 46
argp = 329
lon = 245
to = 2017.5*u.yr
t = 2019.34*u.yr
m1 = 1*u.Msun
m2 = 0.2*u.Msun
mu = c.G*m1*m2
m = m2
kep = mu/m
per = period(sma,m1)
#print(per)

meanmotion = np.sqrt(kep/(sma**3)).to(1/u.s)
meananom = meanmotion*((t-to).to(u.s))

pos, vel, acc = keplerian_to_cartesian(sma,ecc,inc,argp,lon,meananom.va
print('pos',pos)
print('vel',vel)
print('acc',acc)
```

```
pos [ 0.78520989 -4.00711834  2.49057801] AU
vel [10.72701811  4.13036521  8.25981672] km / s
acc [-1.34637308  6.84964031 -4.26121914] km / (s yr)
```

It can also return observables for an array of orbits.

Let's generate 10 trial orbits using the **draw_orbits** function, which draws an array of orbital parameters from priors described in Pearce et al. 2019. SMA and Long of Nodes are fixed at 100. AU and 0 deg respectively as part of the Orbits for the Imaptient procedure (OFTI; Blunt et al. 2017), because draw_orbits was written as part of that procedure. For more, see those papers and the **lofti** python package.

keplerian_to_cartesian returns a 3xN array of observables for each of the N orbits input.

In [16]:

```
m1 = 1*u.Msun
m2 = 0.2*u.Msun
mu = c.G*m1*m2
m = m2
kep = mu/m
obsdate = 2019.34

sma, ecc, inc, argp, lon, orbit_fraction = draw_orbits(10)
meananom = orbit_fraction*2*np.pi

pos, vel, acc = keplerian_to_cartesian(sma*u.au,ecc,inc,argp,lon,meanan
print('pos',pos)
print('vel',vel)
print('acc',acc)
```

```
pos [[ -67.77294331   -1.8393458    19.47052144]
 [ -77.17461462    3.15548293    7.05064809]
 [-143.45286343  -10.82848091  -46.01202379]
 [  81.00668142  -58.68805504  -54.09916137]
 [ 192.09148214  -37.52924613   26.41629592]
 [ -76.37061185   67.60231574   80.46728074]
 [  22.07941865    2.00999668    2.36444801]
 [ -21.46273727  -32.49869347  -49.92689809]
 [ -34.84265082   20.70483422   57.80125155]
 [  69.25666329   15.39825834  -54.4231958 ]] AU
vel [[ 0.01022346  0.37949517 -4.01717216]
 [-2.36871988 -1.18350478 -2.64443698]
 [-0.46631186 -0.37349477 -1.58704166]
 [ 1.87894989  1.31523042  1.21239088]
 [-0.18040457 -0.23106871  0.16264594]
 [-2.05726617 -0.4785954  -0.56967383]
 [-6.42170584  3.51583124  4.13582781]
 [ 1.90350112  2.14953812  3.30227955]
 [-1.49989615 -1.26231379 -3.52397493]
 [ 2.8074744  -0.47850699  1.69122242]] km / s
acc [[ 0.03613682  0.0009807  -0.01038131]
 [ 0.03095409 -0.00126553 -0.00282771]
 [ 0.00779106  0.00058811  0.00249897]
 [-0.01030705  0.00746722  0.00688335]
 [-0.00466644  0.00091169 -0.00064173]
 [ 0.00651822 -0.00576981 -0.00686782]
 [-0.37281028 -0.03395334 -0.03994081]
 [ 0.01582063  0.02395558  0.03680233]
 [ 0.0185337  -0.01101319 -0.0307453 ]
 [-0.01812843 -0.00403055  0.01424545]] km / (s yr)
```

cartesian_to_keplerian takes in the 3D position and velocity array and returns the orbital parameters (as astropy unit objects) that would generate those observables. As of now, it can only handle a single orbit at a time.

Let's take that single orbit from before:

In [22]:

```
sma = 5.2*u.au
ecc = 0.2
inc = 46
argp = 329
lon = 245
to = 2017.5*u.yr
t = 2019.34*u.yr
m1 = 1*u.Msun
m2 = 0.2*u.Msun
mu = c.G*m1*m2
m = m2
kep = mu/m
per = period(sma,m1)

meanmotion = np.sqrt(kep/(sma**3)).to(1/u.s)
meananom = meanmotion*((t-to).to(u.s))
print('mean anomaly:',meananom)

pos, vel, acc = keplerian_to_cartesian(sma,ecc,inc,argp,lon,meananom.va
print('pos',pos)
print('vel',vel)
print('acc',acc)
```

```
mean anomaly: 0.9749547815875452
pos [ 0.78520989 -4.00711834  2.49057801] AU
vel [10.72701811  4.13036521  8.25981672] km / s
acc [-1.34637308  6.84964031 -4.26121914] km / (s yr)
```

And compute the orbital elements:

In [23]:

```
cartesian_to_keplerian(pos,vel,kep)
```

Out[23]:

```
(<Quantity 5.2 AU>,
 <Quantity 0.2>,
 <Quantity 46. deg>,
 <Quantity 329. deg>,
 <Quantity 245. deg>,
 <Quantity 0.9749548>)
```

Looks good!

In [ ]:

```

```