Ferrers Potential in galpy

1. Variables and Units

amp – Amplitude to be applied to the potential (default: 1); can be a Quantity with units of mass or $G \times$ mass
a – Scale radius (can be Quantity)
n – Power of Ferrers density ($n > 0$), not necessarilly integer, calculations generally fail for $n > 4$
b – y-to-x axis ratio of the density, usually $b < a$
c – z-to-x axis ratio of the density, usually $c < b$
Ω_b – pattern speed of the bar, $\vec{\Omega}_b = \hat{e}_z\,\Omega_b$ - not included in the code yet!
pa – If set, the position angle of the bar in the xy plane measured from x (rad or Quantity)
normalize – if True, normalize such that $v_c(1.,0.) = 1.$, or, if given as a number, such that the force is this fraction of the force necessary to make $v_c(1.,0.) = 1.$.
r_0, v_0 – Distance and velocity scales for translation into internal units (default from configuration file)
Variables α appearing in the code in form _α2 carry the value of α^2 initialized in the begining of the class __init__ function.

2. Functions

(... those I have written so far, except for potential evaluating function itself implemented for rotating potential in an inertial frame of reference in which a bar lies aligned with x axis.)

2.1. Evaluate

Purpose: Evaluation of the rotating bar's potential as a function of (R, ϕ, z) in inertial frame taking into account initial position declared by 'pa'.

$$\Phi(\vec{x}) = \frac{-\text{amp}\,b\,c}{4(n+1)}\,\Phi' \tag{1}$$

```
def _evaluate(self,R,z,phi=0.,t=0.):
    if not self.isNonAxi:
        phi= 0.
    x,y,z= bovy_coords.cyl_to_rect(R,phi,z)
    xy= numpy.dot(self.rot(t),numpy.array([x,y]))
    x,y= xy[0],xy[1]
    return self._evaluate_xyz(x,y,z)

def _evaluate_xyz(self,x,y,z=0.):
    return -1/4/(self.n+1)*self._b*self._c*_potInt(x,y,z,self._a2,self._b2,
self._c2,self.n)
```

2.2. X Force

Purpose: Evaluation of the x component of the force as a function of (x,y,z) in the aligned coordinate frame, which is used for evaluation of the force in cylindrical coordinates and then in orbit integration; does not take into account bar's rotation or initial position and therefore shall not be used directly.

$$F_x = \frac{-\text{amp}\,b\,c}{2}\,\Theta_1' \tag{2}$$

```
def _xforce_xyz(self,x,y,z):
    return 1/2*self._b*self._c*_forceInt(x,y,z,self._a2,self._b2,self._c2,0,
self.n)
```

2.3. Y Force

Purpose: Evaluation of the y component of the force as a function of (x,y,z) in the aligned coordinate frame, which is used for evaluation of the force in cylindrical coordinates and then in orbit integration; does not take into account bar's rotation or initial position and therefore shall not be used directly.

$$F_x = \frac{-\mathrm{amp}\, b\, c}{2}\, \Theta_2' \tag{3}$$

```
def _yforce_xyz(self,x,y,z):
    return 1/2*self._b*self._c*_forceInt(x,y,z,self._a2,self._b2,self._c2,1,
self.n)
```

2.4. Z Force

Purpose: Evaluation of the z component of the force as a function of (x,y,z) in the aligned coordinate frame, which is used for evaluation of the force in cylindrical coordinates and then in orbit integration; does not take into account bar's rotation or initial position and therefore shall not be used directly.

$$F_x = \frac{-\mathrm{amp}\, b\, c}{2}\, \Theta_3' \tag{4}$$

```
def _zforce_xyz(self,x,y,z):
    return 1/2*self._b*self._c*_forceInt(x,y,z,self._a2,self._b2,self._c2,2,
self.n)
```

2.5. Density

Purpose: Evaluation of the density as a function of (x,y,z) in the aligned coordinate frame from cylindrical coordinates given as input

$$\rho(m^2) = \begin{cases} \frac{\mathrm{amp}}{4\,\pi\, a^3}\,(1-(m/a)^2)^n, & \text{for } m < a \\ 0, & \text{for } m \geq a \end{cases}$$

where

$$m^2 = x'^2 + \frac{y'^2}{b^2} + \frac{z'^2}{c^2} \tag{5}$$

```
def _dens(self,R,z,phi=0.,t=0.):
    x,y,z= bovy_coords.cyl_to_rect(R,phi,z)
    xy= numpy.dot(self.rot(t),numpy.array([x,y]))
    x,y= xy[0],xy[1]
    m2 = x**2/self._a2+y**2/self._b2+z**2/self._c2
    if m2 < 1:
        return 1/(4*numpy.pi*self.a**3)*(1-m2/self.a**2)**self.n
    else:
        return 0
```

2.6. General Second Derivative

Purpose: General 2nd derivative of the potential as a function of (x,y,z) in the aligned coordinate frame

$$\Phi_{ij} = -\frac{1}{4}\, b\, c\, \Phi'_{ij} \tag{6}$$

```
def _2ndDerivInt(x,y,z,a2,b2,c2,i,j,n):
    def integrand(tau):
        if i!=j:
            return _FracInt(x,y,z,a2,b2,c2,tau,n-1)*n*(1+(-1-2*x/(tau+a2))*(i==0
     or j==0))*(1+(-1-2*y/(tau+a2*b2))*(i==1 or j==1))*(1+(-1-2*z/(tau+a2*c2))*(
    i==2 or j==2))
        else:
            var2 = x**2*(i==0) + y**2*(i==1) + z**2*(i==2)
            coef2 = a2*(i==0) + a2*b2*(i==1) + a2*c2*(i==2)
            return _FracInt(x,y,z,a2,b2,c2,tau,n-1)*n*(4*var2)/(tau+coef2)**2 +
    _FracInt(x,y,z,a2,b2,c2,tau,n)*(-2/(tau+coef2))
    return integrate.quad(integrand,lowerlim(x,y,z,a2,b2,c2),numpy.inf)[0]
```

2.7. Integration for Potential

Purpose: Evaluation of the z component of the force as a function of (x,y,z) in the aligned coordinate frame

$$\Phi' = \int_{\lambda}^{\infty} A^{n+1}(\tau)\, \mathrm{d}\tau \tag{7}$$

```
def _potInt(x,y,z,a2,b2,c2,n):
    def integrand(tau):
        return _FracInt(x,y,z,a2,b2,c2,tau,n + 1)
    return integrate.quad(integrand,lowerlim(x,y,z,a2,b2,c2),numpy.inf)[0]
```

2.8. Integration for Forces

Purpose: Evaluation of the z component of the force as a function of (x,y,z) in the aligned coordinate frame

$$\Theta'_i = \int_{\lambda}^{\infty} \frac{x_i}{a_i^2 + \tau} A^n(\tau)\, \mathrm{d}\tau, \tag{8}$$

where

$$a_1 = a,\ a_2 = ab,\ a_3 = ac \tag{9}$$

```
def _forceInt(x,y,z,a2,b2,c2,i,n):
    def integrand(tau):
        return -(x*(i==0) + y*(i==1) + z*(i==2))/(a2*(i==0) + a2*b2*(i==1) + a2*
    c2*(i==2) + tau)*_FracInt(x,y,z,a2,b2,c2,tau,n)
    return integrate.quad(integrand,lowerlim(x,y,z,a2,b2,c2),numpy.inf)[0]
```

2.9. Integration for Second Derivative

Purpose: Integral that gives the 2nd derivative of the potential in x,y,z

The derivative is generally $\frac{\partial^2 \Phi}{\partial x_i \partial x_j}$; for $i == j$ the integral to be evaluated is:

$$\Phi'_{ii} = \int_{\lambda}^{\infty} \frac{4\,n\,x_i^2}{(\tau + a_i^2)^2} \frac{\left(1 - \sum_{i=1}^{3} \frac{x_i^2}{\tau + a_i^2}\right)^{n-1}}{[(\tau + a^2)](\tau + a^2 b^2)(\tau + a^2 c^c)]^{\frac{1}{2}}} - \frac{2}{\tau + a_i^2} \frac{\left(1 - \sum_{i=1}^{3} \frac{x_i^2}{\tau + a_i^2}\right)^{n}}{[(\tau + a^2)](\tau + a^2 b^2)(\tau + a^2 c^c)]^{\frac{1}{2}}} \mathrm{d}\,\tau \tag{10}$$

In all other cases, the integral has this form:

$$\Phi'_{ij} = \int_{\lambda}^{\infty} \frac{4\,n\,x_i\,x_j}{(\tau + a_i^2)(\tau + a_j^2)} \frac{\left(1 - \sum_{i=1}^{3} \frac{x_i^2}{\tau + a_i^2}\right)^{n-1}}{[(\tau + a^2)](\tau + a^2 b^2)(\tau + a^2 c^c)]^{\frac{1}{2}}} \mathrm{d}\tau \tag{11}$$

```
def _2ndDerivInt(x,y,z,a2,b2,c2,i,j,n):
    def integrand(tau):
        if i!=j:
            return _FracInt(x,y,z,a2,b2,c2,tau,n-1)*n*(1+(-1-2*x/(tau+a2))*(i==0
     or j==0))*(1+(-1-2*y/(tau+a2*b2))*(i==1 or j==1))*(1+(-1-2*z/(tau+a2*c2))*(
    i==2 or j==2))
        else:
            var2 = x**2*(i==0) + y**2*(i==1) + z**2*(i==2)
            coef2 = a2*(i==0) + a2*b2*(i==1) + a2*c2*(i==2)
            return _FracInt(x,y,z,a2,b2,c2,tau,n-1)*n*(4*var2)/(tau+coef2)**2 +
    _FracInt(x,y,z,a2,b2,c2,tau,n)*(-2/(tau+coef2))
    return integrate.quad(integrand,lowerlim(x,y,z,a2,b2,c2),numpy.inf)[0]
```

2.10. Part of Integrand Used in All Functions

Purpose: Returns part of an integrand which is used in other functions so the code was more concise.

$$A^{\nu}(\tau) = \frac{\left(1 - \sum_{i=1}^{3} \frac{x_i^2}{\tau + a_i^2}\right)^{\nu}}{[(\tau + a^2)(\tau + a^2 b^2)(\tau + a^2 c^2)]^{\frac{1}{2}}} \tag{12}$$

```
def _FracInt(x,y,z,a2,b2,c2,tau,expon):
    return (1 - x**2/(a2 + tau) - y**2/(a2*b2 + tau) - z**2/(a2*c2 + tau))**
    expon/numpy.sqrt((a2 + tau)*(a2*b2 + tau)*(a2*c2 + tau))
```

2.11. Lower limit for the integrals

Purpose: Evaluates lower limit for integrals used in other functions, based on definition of the density distribution.

$$\lambda = \begin{cases} \text{unique positive solution of } m^2(\lambda) = 1, & \text{for } m \geq 1 \\ \lambda = 0\,, & \text{for } m < 1 \end{cases}$$

where $m^2(\lambda) = \sum_{i=1}^{3} \frac{x_i^2}{\lambda + a_i^2}$.

```
def lowerlim(x,y,z,a2,b2,c2):
    def func(tau):
        return x**2/(a2+tau) + y**2/(a2*b2+tau) + z**2/(a2*c2+tau) - 1

    if numpy.sqrt(x**2/a2 + y**2/(a2*b2) + z**2/(a2*c2)) >= 1:
        return fsolve(func,0)[0]
    else:
        return 0
```