Imports

In [2]:
```python
import numpy as np
import numba
import matplotlib.pyplot as plt
%config InlineBackend.figure_format = 'retina'
plt.rcParams["figure.figsize"] = (5,5)

@numba.njit
def rk4(f,dt,u0,t0):
    k1 = dt*f(u0,t0)
    k2 = dt*f(u0+0.5*k1,t0+0.5*dt)
    k3 = dt*f(u0+0.5*k2,t0+0.5*dt)
    k4 = dt*f(u0+k3,t0+dt)
    return u0+(k1+2*k2+2*k3+k4)/6.0

def evolve(f,t0,t1,N,u0,step):
    T,dt = np.linspace(t0,t1,N,retstep=True)
    U = np.array([u0 for n in range(0,N)])
    for n in range(0,N-1):
        U[n+1] = step(f,dt,U[n],T[n])
    return T, U
```

Jupiter's Orbit

Recall that an elliptical orbit (aligned with the x and y axes) can be written in the following form

$$r = \frac{a(1-e^2)}{1+e\cos\theta}$$

where a is the semi-major eaxis of the orbit and e is it's eccentricity. Alternatively one can use the perihelion (minimal, at $\theta = 0$) and aphelion (maximal, at $\theta = \pi$) distances r_p and r_a to characterize the orbit with

$$a = \frac{r_a + r_p}{2} \qquad e = \frac{r_a - r_p}{r_a + r_p} \tag{1}$$

or in reverse $r_a = (1+e)a$ and $r_p = (1-e)a$. The time-dependence of θ is somewhat non-trivial and can be determined from the equations

$$\begin{aligned} t &= \frac{T}{2\pi}(E - e\sin E) \\ r &= a(1-e\cos E) \\ \tan(\theta/2) &= \sqrt{(1+e)/(1-e)}\tan(E/2) \end{aligned}$$

These equations involve solving a non-linear equation at each time for the quantity E. Since we'd rather not do that at each step of integration routine, what we'll use our ODE solver to integrate Jupiter's trajectory. Since our `rk4` method needs a continuous function of time, we'll write a short routine to take that computes trajectory over one period on a fine time-grid and interpolate to give values at arbitrary times.

For Jupiter one has the parameters $a_J \approx 7.7857 \cdot 10^8$ km = 5.2044 AU. It's perihelion and aphelion are given by $r_{p,J} = 4.9501$ AU and $r_{a,J} = 5.4588$ AU (respectively) yielding a small eccentricity of $\epsilon_J \approx 0.0489$. Explicitly, we consider an initial position of the form

$$\vec{r}_J(0) = r_{J,a}\hat{x}$$

At the aphelion the velocity is perpendicular to $\vec{\rho}$ so we know the direction is along $\hat{y}$. Its magnitude is fixed by the relation

$$|\vec{v}_J(0)| = \sqrt{\frac{Gm_S}{a_J}\left(\frac{1-e_J}{1+e_J}\right)}$$

With these initial conditions we can compute $\vec{r}_J(t)$ as a function of time using our usual methods. The equation of motion is then

$$\frac{d^2\vec{r_J}(t)}{dt^2} = -GM\left(\frac{\vec{r_J}(t)}{|\vec{r_J}(t)|^3}\right)$$

To simplify our notation we will use the astromomical unit 1 AU as our length and 1 year (365 days) as our time unit. In these units the dimensional constant GM is given by $GM/(1\ \mathrm{AU})^3 \cdot (1\ \mathrm{year})^2 = (2\pi)^2 \equiv \gamma_S$. In these units the initial conditions read

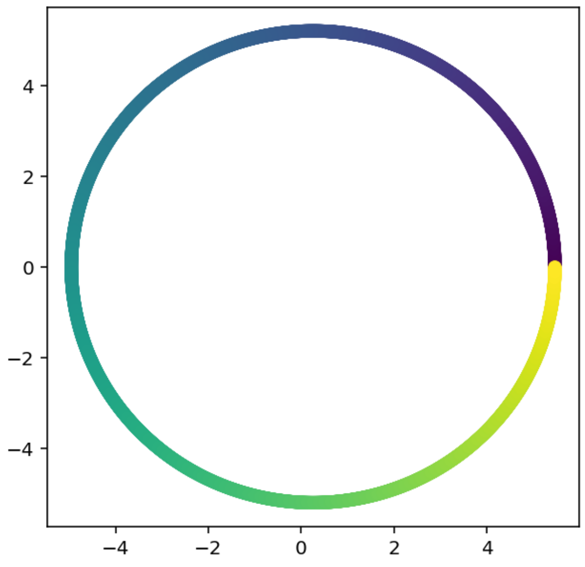
$$\vec{r}_J(0) = 5.4588\hat{x}\ \mathrm{AU}$$
$$\vec{v}_J(0) = 2.621\hat{y}\ \mathrm{AU/year}$$

where we've substituted the values of G, M, a_J and e_J to get the magnitude of the velocity.

In [3]:

```
# gravitational parameter in natural units for the Sun
γS = (2.0*np.pi)**2

@numba.njit
def sun(u,τ): # acceleration from the sun alone
    rx,ry,vx,vy = u
    r = np.array([rx,ry])
    nr = np.linalg.norm(r)
    ax,ay = -γS*r/nr**3
    return np.array([vx,vy,ax,ay])

# useful units of time:
years = 1.0
days  = 1.0/365

# Jupiter's period
jupiter_period = 11.87*years

# choice of time step
jupiter_dt = 1*days
# number of steps
jupiter_nt = int(jupiter_period/jupiter_dt)

# jupiter's orbital parameters
jupiter_a  = 5.2044
jupiter_e  = 0.0489
jupiter_ra = jupiter_a*(1+jupiter_e)
jupiter_rp = jupiter_a*(1-jupiter_e)
jupiter_v0 = np.sqrt(γS/jupiter_a*(1-jupiter_e)/(1+jupiter_e))

# initial condition for Jupiter
jupiter_u0 = np.array([jupiter_ra,0.0,
                       0.0,jupiter_v0])
jupiter_T,jupiter_U = evolve(sun,
                             0.0,jupiter_period,
                                    jupiter_nt,
                                    jupiter_u0,step=rk4)

# get the positions
jupiter_rx,jupiter_ry = jupiter_U[:,0],jupiter_U[:,1]

# plot the positions, with time as color to verify that
# (a) the orbit is nearly circular and (b) it's executed
# one orbit in the time window
plt.scatter(jupiter_rx,jupiter_ry,c=jupiter_T)
```

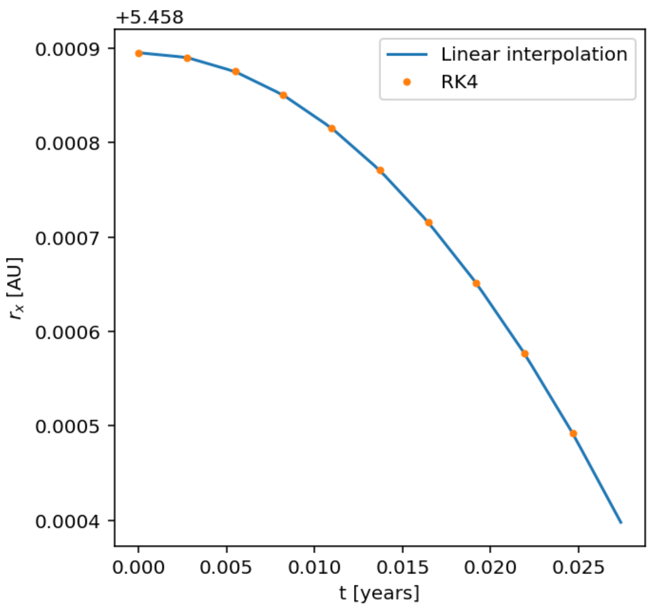
Out[3]:
```
<matplotlib.collections.PathCollection at 0x1aa96b3b610>
```



We'll now construct a function that for any arbitrary time t uses the arrays `jupiter_rx` and `jupiter_ry` to interpolate to find the (approximate) position of Jupiter.

In [4]:
```
@numba.njit
def jupiter_r(t):
    # remove multiples of period
    tr = np.mod(t,jupiter_period)
    # Get nearest two indices in the jupiter_rx, jupiter_ry arrays
    # floor gives the nearest below, use % to wrap at the
    # edges. We've assumed that t>0.
    tx = tr/jupiter_period*jupiter_nt
    itL = round(np.floor(tx))%jupiter_nt
    itR = (itL+1)%jupiter_nt
    # get those two times as well
    tL  = jupiter_T[itL]
    tR  = tL+jupiter_dt
    # and the two positions at those time indices
    rL = np.array([jupiter_rx[itL],jupiter_ry[itL]])
    rR = np.array([jupiter_rx[itR],jupiter_ry[itR]])

    # interpolate linearly between those points
    return ((tR-tr)*rL + (tr-tL)*rR)/(tR-tL)


# check that this works: look at rx for the window
# [0,10*dt] using the interpolant and the stored values
T = np.linspace(0,10*jupiter_dt)
plt.plot(T,[jupiter_r(t)[0] for t in T],label='Linear interpolation')
plt.plot(jupiter_T[:10],jupiter_rx[:10],marker='.',lw=0,label='RK4')
plt.ylabel('$r_x$ [AU]'); plt.xlabel('t [years]'); plt.legend()
```

```
Out[4]: <matplotlib.legend.Legend at 0x1aa98025e50>
```



Dynamics in the asteroid belt

The gravitational acceleration for a small asteroid with position $\vec{r}(t)$ that is affected by Jupiter and the Sun along then takes the form

$$\frac{d^2\vec{r}(t)}{dt^2} = -GM_J\left(\frac{\vec{r}(t)-\vec{r}_J(t)}{|\vec{r}(t)-\vec{r}_J(t)|^3}\right) - GM\left(\frac{\vec{r}(t)}{|\vec{r}(t)|^3}\right)$$

where $\vec{r}_J(t)$ is determined numerically as above. In our natural units we note that the constant in front of the term from Jupiter GM_J is given by $GM_J/(1\text{ AU})^3 \cdot (365\text{ days})^2 = (2\pi)^2 M_J/M \approx 0.03763 \equiv \gamma_J$.

We can convert this into a routine that will work with our non-symplectic solvers easily, following what we did in Assignment #2.

```
In [5]:
# gravitational parameters for Jupiter and the Sun
γJ = γS*0.9546e-3

# equations of motion from the gravity of the Sun and Jupiter
@numba.njit
def sun_jupiter(u,t):
    rx,ry,vx,vy = u
    rS  = np.array([rx,ry])
    nrS = np.linalg.norm(rS)
    rJ  = rS - jupiter_r(t)
    nrJ = np.linalg.norm(rJ)
    ax,ay = -γS*rS/nrS**3-γJ*rJ/nrJ**3
    return np.array([vx,vy,ax,ay])
```

Generating Asteroid Belt Orbits

We now need to look at the asteroid belt itself. What kind of initial conditions should we consider? For simplicity we will ignore the inclination of their orbits and assume they are all in the same plane as the planets. When intializing the orbits we'll also ignore the effects of Jupiter, and discuss the orbital properties arising from the Sun alone.

For the most part known asteroids lie in orbits with distances from the Sun in the range of 2 to 3.5 AU. We will thus consider orbits such that the minimum and maximum distance (the perihelion and aphelion) lie in this range. In our natural units this is $r_{\rm min} \equiv 2.0{\rm AU} < |\vec{r}| < 3.5{\rm AU} \equiv r_{\rm max}$, i.e. well inside Jupiter's orbit. Since we're looking at these after many orbits, we'll always start the object at the aphelion for simplicity, using the same equations we used for Jupiter.

Explicitly, we consider an initial position of the form

$$\vec{r} = r_a\left(\cos\theta\hat{x} + \sin\theta\hat{y}\right)$$

where θ characterizes the tilt of the orbit away from the $\hat{x}$ axis and r_a is the aphelion distance. At the aphelion the velocity is perpendicular to $\vec{r}$ so we know the direction is along $-\sin\theta\hat{x} + \cos\theta\hat{y}$. Its magnitude is fixed by

$$|\vec{v}| = \sqrt{\frac{Gm_S}{a}\left(\frac{1-e}{1+e}\right)}$$

where $a = (r_a + r_p)/2$ and $e = (r_a - r_p)/(r_a + r_p)$. Our description of the asteroid belt will thus involve orbits with r_a and r_p in that range, with arbitrary values for the angle θ.

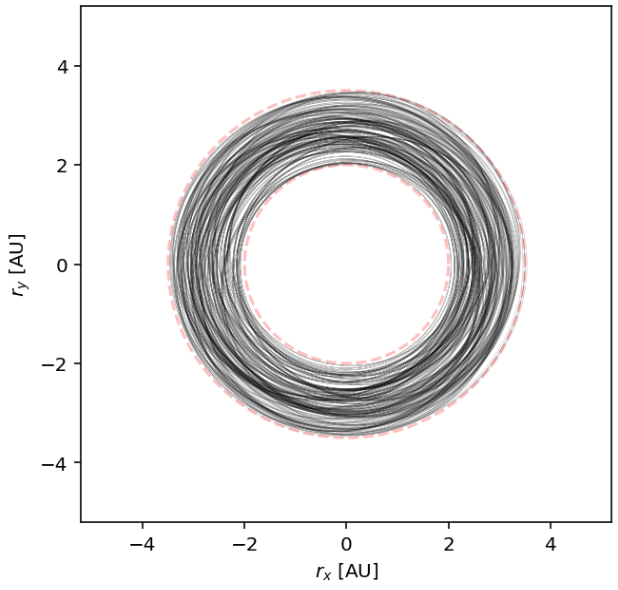
We will also allow for a "delay time" t_0 that evolves the asteroid for some time to start it away from its aphelion. We can do this simply using the same `rk4` routine we used for Jupiter, keeping only the Sun's gravity.

In [21]:
```python
# some quality of life improvements: functions that encode known
# relationships between positions and velocities and the orbtial
# parameters (see text)
@numba.njit
def eccentricity(u): # from position & velocity
    rx,ry,vx,vy = u
    h = rx*vy - ry*vx
    en  = 0.5*(vx**2+vy**2) -γS/np.sqrt(rx**2+ry**2)
    return np.sqrt(1 + 2*en*(h/γS)**2)

@numba.njit
def semimajor_axis(u): # from position & velocity
    rx,ry,vx,vy = u
    en  = 0.5*(vx**2+vy**2) -γS/np.sqrt(rx**2+ry**2)
    return -γS/(2*en)

@numba.njit
def period(u): # from position & velocity
    a = semimajor_axis(u)
    return 2.0*np.pi*np.sqrt(a**3/γS)

# distance from the Sun at edges of asteroid belt
rmin = 2.0 # AU
rmax = 3.5 # AU

# generate initial conditions for an elliptical orbit
# with perihelion rp and aphelion ra starting at the
# aphelion at angle ϑ
def elliptical_orbit(rp,ra,θ,t0=0.0):
    a = (ra+rp)/2.0 # semi-major axis
    e = (ra-rp)/(ra+rp) # eccentricity

    # the initial position at the aphelion
    r  = np.array([ra*np.cos(θ),ra*np.sin(θ)])
    nr = np.linalg.norm(r)

    # magnitude of the velocity (in natural units)
    nv = np.sqrt(γS/a*(1-e)/(1+e))

    # compute the velocity with the given norm
    v  = np.array([-np.sin(θ),np.cos(θ)])*nv

    # the result in a form that evolve will understand
    u0 = np.array([r[0],r[1],v[0],v[1]])

    # evolve it for the delay time t0, if that time is > 0
    if (t0!=0.0):
        T, U = evolve(sun,0.0,t0,int(t0/days),u0,step=rk4)
        u1 = U[-1,:]
    else:
        u1 = u0

    # return the position and velocity
    return u1

# an alternative form that takes semimajor axis and eccentricity
# rather than perihelion and aphelion
def elliptical_orbit_alt(a,e,θ,t0=0):
    ra = (1+e)*a
    rp = (1-e)*a
    return elliptical_orbit(rp,ra,θ,t0=t0)

# generate an asteroid belt orbit; we'll keep the range
# to be changeable, but will use the true values as default
# - Rx,Ry drawn uniformly from [Rmin,Rmax]
# - ra,rp = max(Rx,Ry), min(Rx,Ry)
# - ϑ0 drawn uniformly from [0,2π]
# - t0 drawn uniformly from [0,T]
def asteroid_orbit(Rmin=rmin,Rmax=rmax):
    # give a random number for Rx,Ry in range [Rmin,Rmax]
    Rx,Ry = Rmin + (Rmax-Rmin)*np.random.rand(2)
    rp,ra =  min(Rx,Ry), max(Rx,Ry) # min/max for peri/aphelion
    θ  = np.random.rand()*2*np.pi # an initial angle
    # orbit without any delay
    u0 = elliptical_orbit(rp,ra,θ)
    # a random initial delay
    t0 = period(u0)*np.random.rand()
    # return the delayed orbit
    return elliptical_orbit(rp,ra,θ,t0=t0)

def asteroid_orbit_circular(R):
    θ = np.random.rand()*2*np.pi # an initial angle
    return elliptical_orbit(R,R,θ)
```

We need to test this to make sure it's properly generating these orbits.

In [22]:
```
# we fix the seed so the output is reproducible
np.random.seed(0)

dt = 3*days             # choice of time step
t1 = jupiter_period*2   # long enough for a typical orbit or two
nt = int(t1/dt)         # number of steps for this dt, t1
```

```
plt.xlim(-jupiter_a,jupiter_a)
plt.ylim(-jupiter_a,jupiter_a)

# show the boundary of the desired region [0.385,0.673]
θ = np.linspace(0,2*np.pi,300)
plt.plot(rmin*np.cos(θ),rmin*np.sin(θ),color='r',alpha=0.25,ls='dashed')
plt.plot(rmax*np.cos(θ),rmax*np.sin(θ),color='r',alpha=0.25,ls='dashed')

# initialize a large number of random orbits and evolve them
for k in range(0,100):

    u0 = asteroid_orbit() # random orbit in range
    T,U = evolve(sun_jupiter,0.0,t1,nt,u0,step=rk4)

    # make a plot to see that they stay roughly inside the region
    rx,ry = U[:,0],U[:,1]
    plt.plot(rx,ry,lw=0.25,color='k',alpha=0.25)
plt.xlabel('$r_x$ [AU]')
plt.ylabel('$r_y$ [AU]');
plt.rcParams["figure.figsize"] = (5,5);
```



Resonances

First we'll explore the gaps themselves, focusing on orbits that are absent in the distribution of the asteroids. Our goal will be to see how the orbits become destablized; in particular we want to know how long it takes for the effects of Jupiter to push such an asteroid out of the belt.

The first and most prominent resonance is when the asteroid's period is 3 times is three times that of Jupiter. Due to Kepler's square-cube law, this corresponds to a semi-major axis that is $1/3^{2/3} \approx 0.48075$ times smaller. We'll start with orbits with the period, focussing on one with a typical eccentricity $e = 0.15$, evolving for a hundred orbits or so (1000 years). For comparison we'll do the same for an orbit whose period is an irrational multiple of Jupiter's period so no resonance is possible.

In [128…

```
def resonant_orbit(ratio,      # ratio between period of Jupiter and asteroid
                   t1,         # time period to simulate for
                   t0 = 0,     # initial time-delay on asteroid
                   e = 0,      # eccentricity off the initial orbit
                   θ = 0,      # angle of aphelion relative to Jupiter at t=0
                   dt = 3*days): # choice of time step
```

```
    a      = jupiter_a/pow(ratio,2.0/3.0)
    period = jupiter_period/ratio

    nt = int(t1/dt) # number of steps for this dt, t1

    rp,ra = a*(1-e),a*(1+e) # peri- and aphelion of orbit
    u0 = elliptical_orbit(rp,ra,θ,t0=t0) # intial condition

    # starting from the new position, evolve for the longer time period
    T,U = evolve(sun_jupiter,0.0,t1,nt,u0,step=rk4)

    # compute semi-major axis locally at each time
    A = np.array([semimajor_axis(U[it,:]) for it in range(0,nt)])

    # compute the eccentricity locally at each time
    E = np.array([eccentricity(U[it,:]) for it in range(0,nt)])

    return T,A,E
```

Looking at a resonant and non-resonant orbit

In [32]:

```
%time T1,A1,E1 = resonant_orbit(2*np.sqrt(2),10000*years,e=0.15,θ=0*np.pi)
%time T2,A2,E2 = resonant_orbit(3.00,        10000*years,e=0.15,θ=0*np.pi)
```

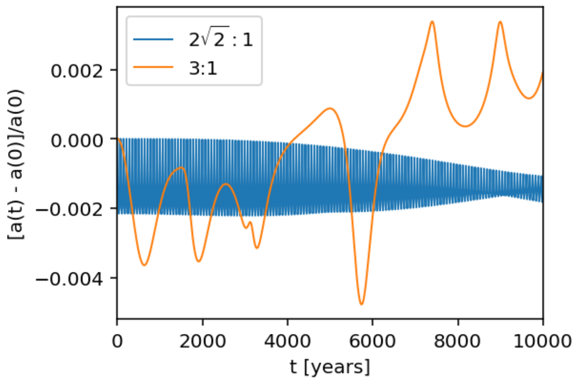
```
Wall time: 22.2 s
Wall time: 22.1 s
```

In [49]:

```
plt.rcParams["figure.figsize"] = (4,3);
t1=10000*years
# since we have many, many points, don't plot
# all of them, only plot every tskip^th point
tskip = int(jupiter_period/(3*days)) # line up skip rate with Jupiter's period
plt.plot(T1[::tskip],(A1[::tskip]-A1[0])/A1[0],lw=1,label="$2\sqrt{2}:1$")
plt.plot(T2[::tskip],(A2[::tskip]-A2[0])/A2[0],lw=1,label='3:1')
plt.xlabel('t [years]')
plt.ylabel('[a(t) - a(0)]/a(0)')
plt.xlim(0,t1)
plt.legend()
```

Out[49]:

```
<matplotlib.legend.Legend at 0x1aa98517a30>
```

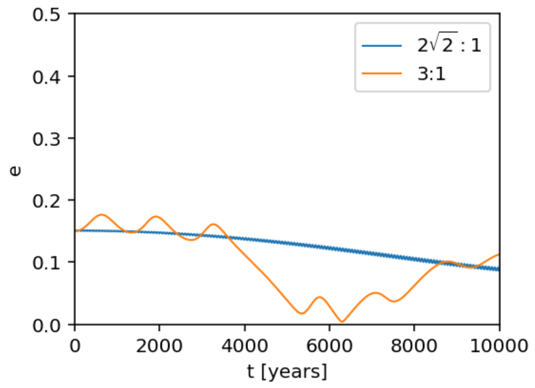


In [54]:

```
plt.plot(T1[::tskip],E1[::tskip],lw=1,label="$2\sqrt{2}:1$")
plt.plot(T2[::tskip],E2[::tskip],lw=1,label='3:1')
plt.xlabel('t [years]')
plt.ylabel('e')
plt.ylim(0,0.5)
plt.xlim(0,t1)
plt.legend()
```

Out[54]:

```
<matplotlib.legend.Legend at 0x1aa99b7e8e0>
```



Characterizing changes in semimajor axis for circular orbits

We'll now see how this holds up for a variety of initial a. We'll do this for circular orbits, and will characterize the deviation by the "width" of the distribution of $a(t)$. We'll define this width as $\max_t|a(t)-a_{\rm avg}|/a_{\rm avg}$ i.e. where $a_{\rm avg}$ is the time-average of $a(t)$. This will give us the *variability* of a. To get an idea of the total change, we'll also look at $a_{\rm avg}/a(0)$ directly.

In [63]:

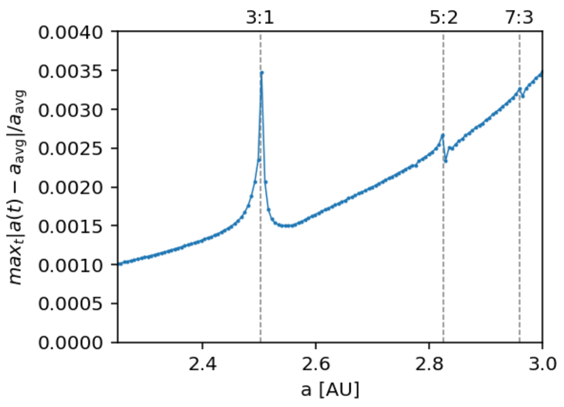
```
%%time
# some lists to store the output
A_avg,A_wid,R = [],[],[]
# loop through radii for circular orbits in the belt; zoom in
# on the region near the 3:1, 5:2, 7:3 resonances
for a in np.linspace(2.25,3.0,128):
    ratio = (jupiter_a/a)**(1.5) # the period ratio for this radius
    # generate a resonant orbit, evolve for 1000 years
    T,A,E = resonant_orbit(ratio,1000*years,e=0.0,θ=0.0*np.pi)
    a_avg = np.mean(A)                       # compute the average semimajor axis
    a_wid = np.max(np.abs(A-a_avg)/a_avg) # .. and the variability
    # add these quantities to our lists
    R.append(ratio)
    A_wid.append(a_wid)
    A_avg.append(a_avg)
```

```
Wall time: 4min 43s
```

Let's plot this as a function of the semimajor axis, highlighting the location of the low order integer ratios.

In [67]:

```
for (n,d) in [(3,1),(5,2),(7,3)]:
    f = float(n)/float(d)
    a = jupiter_a/f**(2.0/3.0)
    plt.plot([a,a],[0,0.004],color='gray',lw=0.75,ls='dashed')
    plt.text(a,0.00405,str(n)+':'+str(d),color='k',va='bottom',ha='center')
R=np.array(R)
plt.ylim(0,0.004)
plt.xlim(2.25,3.0)
plt.plot(jupiter_a/R**(2/3),A_wid,marker='.',lw=0.75,ms=2)
plt.xlabel('a [AU]')
plt.ylabel(r'$max_{t}|a(t)-a_{\rm avg}|/a_{\rm avg}$');
```



To get a broader picture, repeat the same calculation over the full range. We justify this post-hocly; the resonance at 2:1 dominates the plot (as expected) so it helps to have two separate ranges. We'll use it also as an opportunity to plot the average $a$ itself rather than the variability.

In [68]:

```
%%time
# some lists to store the output
A_avg,A_wid,R = [],[],[]
# loop through radii for circular orbits over the full range of the belt
for a in np.linspace(2.0,3.5,128):
    ratio = (jupiter_a/a)**(1.5) # the period ratio for this radius
    # generate a resonant orbit, evolve for 1000 years
    T,A,E = resonant_orbit(ratio,1000*years,e=0.0,θ=0.0*np.pi)
    a_avg = np.mean(A)                     # compute the average semimajor axis
    a_wid = np.max(np.abs(A-a_avg)/a_avg) # .. and the variability
    # add these quantities to our lists
    R.append(ratio)
    A_wid.append(a_wid)
    A_avg.append(a_avg)
```

```
Wall time: 4min 43s
```

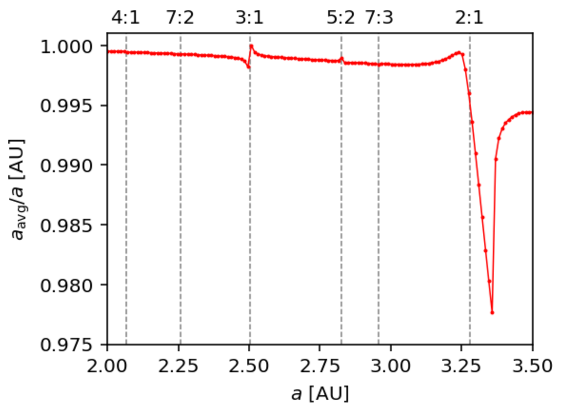
Basically the same plot, buit now for $a_{\rm avg}/a(0)$.

In [69]:

```
for (n,d) in [(2,1),(7,2),(4,1),(3,1),(5,2),(7,3)]:
    f = float(n)/float(d)
    a = jupiter_a/f**(2.0/3.0)
    plt.plot([a,a],[0.975,1.001],color='gray',lw=0.75,ls='dashed')
    plt.text(a,1.0015,str(n)+':'+str(d),color='k',va='bottom',ha='center')
R=np.array(R)
A_avg=np.array(A_avg)
plt.xlim(2.0,3.5)
plt.ylim(0.975,1.001)
A0 = jupiter_a/R**(2/3)
plt.plot(A0,A_avg/A0,marker='.',lw=0.75,ms=2,color='r')
plt.xlabel('$a$ [AU]')
plt.ylabel(r'$a_{\rm avg}/a$ [AU]')
```

Out[69]:

```
Text(0, 0.5, '$a_{\\rm avg}/a$ [AU]')
```



Kirkwood Gaps

We now long at histograms for the three ensembles defined in the report.

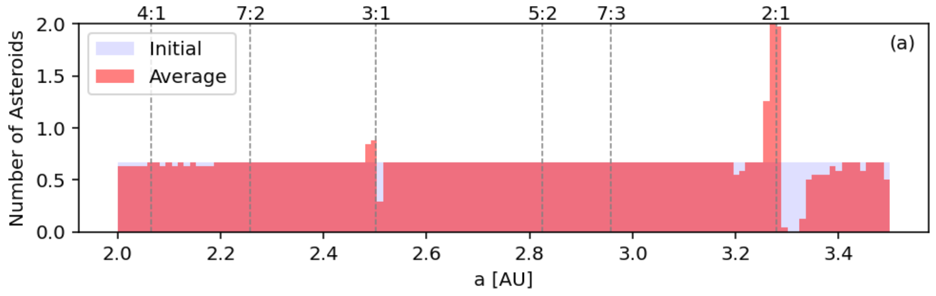
Circular orbits, varying delay time

In [84]:
```
%%time

dt = 10*days
t1 = 1000*years
nt = int(t1/dt)

A1,E1 = [],[] # store values of semi-major axis and eccentricity

# initialize a large number of orbits, sweeping the radius
# and doing an array of different initial locations relative to
# Jupiter (varying the "delay time")
for a in np.linspace(rmin,rmax,128):
    for f in np.linspace(0.0,1.0,16):

        period = 2.0*np.sqrt(a**3/γS) # period of the orbit
        t0 = f*period # delay time

        u0 = elliptical_orbit_alt(a,0.0,θ=0.0,t0=t0)  # circular orbit, with delay t0

        # evolve it for an orbit or so, both Sun and Jupiter
        T,U = evolve(sun_jupiter,0.0,t1,nt,u0,step=rk4)

        # store the averaged orbital parameters
        A1.append(np.mean([semimajor_axis(U[it,:]) for it in range(0,nt)]))
        E1.append(np.mean([eccentricity(U[it,:]) for it in range(0,nt)]))

A1  = np.array(A1);
E1  = np.array(E1);
```

```
Wall time: 22min 17s
```

In [98]:
```
plt.rcParams["figure.figsize"] = (8,2);
ymax = 2.0
for (n,d) in [(3,1),(5,2),(7,2),(7,3),(2,1),(4,1)]:
    f = float(n)/float(d)
    a = jupiter_a/f**(2.0/3.0)
    plt.plot([a,a],[0,ymax],color='gray',lw=0.75,ls='dashed')
    plt.text(a,ymax,str(n)+':'+str(d),color='k',va='bottom',ha='center')

plt.hist(np.linspace(rmin,rmax,128), bins=128,alpha=0.125,
         range=(rmin,rmax),density=True,color='b',label='Initial');
```

```
plt.hist(A1, bins=128,alpha=0.50,range=(rmin,rmax),density=True,color='r',label='Average');
plt.legend(loc='upper left')
plt.ylim(0,ymax)
plt.text(3.55,1.9,'(a)',va='top',ha='right')
plt.xlabel('a [AU]');
plt.ylabel('Number of Asteroids');
```

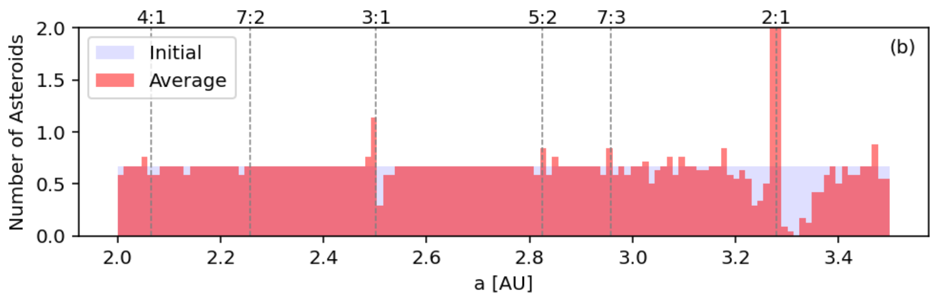


Elliptical orbits, varying the angle relative to Jupiter

In [92]:
```
%%time

dt = 10*days
t1 = 1000*years
nt = int(t1/dt)

A2,E2 = [],[] # store values of semi-major axis and eccentricity

# initialize a large number of orbits, sweeping the radius
# and doing an array of different initial locations relative to
# Jupiter (varying the "delay time")
for a in np.linspace(rmin,rmax,128):
    for θ in np.linspace(0.0,2*np.pi,16):
        e  = 0.15 # fixed eccentricity
        u0 = elliptical_orbit_alt(a,e,θ=θ)  # elliptical orbit, vary ϑ

        # evolve it for an orbit or so, both Sun and Jupiter
        T,U = evolve(sun_jupiter,0.0,t1,nt,u0,step=rk4)

        # store the averaged orbital parameters
        A2.append(np.mean([semimajor_axis(U[it,:]) for it in range(0,nt)]))
        E2.append(np.mean([eccentricity(U[it,:]) for it in range(0,nt)]))

A2  = np.array(A2);
E2  = np.array(E2);
```

```
Wall time: 22min 18s
```

In [97]:
```
plt.rcParams["figure.figsize"] = (8,2);
ymax = 2.0
for (n,d) in [(3,1),(5,2),(7,2),(7,3),(2,1),(4,1)]:
    f = float(n)/float(d)
    a = jupiter_a/f**(2.0/3.0)
    plt.plot([a,a],[0,ymax],color='gray',lw=0.75,ls='dashed')
    plt.text(a,ymax,str(n)+':'+str(d),color='k',va='bottom',ha='center')

plt.hist(np.linspace(rmin,rmax,128), bins=128,alpha=0.125,
         range=(rmin,rmax),density=True,color='b',label='Initial');
plt.hist(A2, bins=128,alpha=0.50,range=(rmin,rmax),density=True,color='r',label='Average');
plt.legend(loc='upper left')
plt.ylim(0,ymax)
plt.xlabel('a [AU]');
plt.text(3.55,1.9,'(b)',va='top',ha='right')
plt.ylabel('Number of Asteroids');
```



Elliptical orbits, random delay time, random angle relative to Jupiter

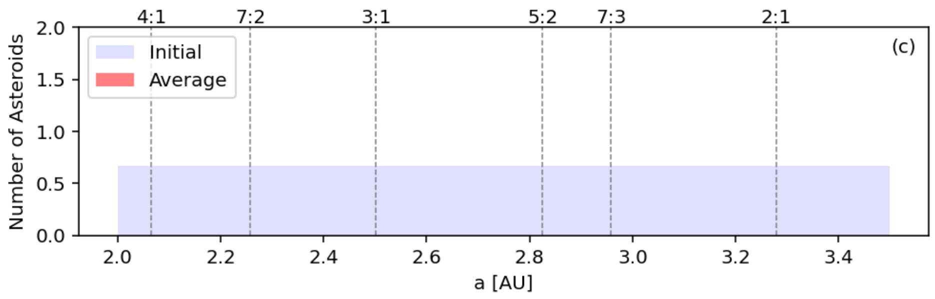
In [ ]:

```
%%time

dt = 10*days
t1 = 1000*years
nt = int(t1/dt)

A3,E3 = [],[] # store values of semi-major axis and eccentricity

# initialize a large number of orbits, sweeping the radius
# and doing an array of different initial locations relative to
# Jupiter (varying the "delay time")
for a in np.linspace(rmin,rmax,128):
    for f in np.linspace(0.0,1.0,16):

        period = 2.0*np.sqrt(a**3/γS) # period of the orbit
        t0 = 1*years+np.random.rand()*period
        θ  = np.random.rand()*2*np.pi
        e=0.15

        u0 = elliptical_orbit_alt(a,e,θ=θ,t0=t0)  # circular orbit, with delay t0

        # evolve it for an orbit or so, both Sun and Jupiter
        T,U = evolve(sun_jupiter,0.0,t1,nt,u0,step=rk4)

        # store the averaged orbital parameters
        A3.append(np.mean([semimajor_axis(U[it,:]) for it in range(0,nt)]))
        E3.append(np.mean([eccentricity(U[it,:]) for it in range(0,nt)]))

A3  = np.array(A3);
E3  = np.array(E3);
```

In [233…

```
plt.rcParams["figure.figsize"] = (8,2);
ymax = 2.0

for (n,d) in [(3,1),(5,2),(7,2),(7,3),(2,1),(4,1)]:
    f = float(n)/float(d)
    a = jupiter_a/f**(2.0/3.0)
    plt.plot([a,a],[0,ymax],color='gray',lw=0.75,ls='dashed')
    plt.text(a,ymax,str(n)+':'+str(d),color='k',va='bottom',ha='center')

plt.hist(np.linspace(rmin,rmax,128), bins=128,alpha=0.125,
         range=(rmin,rmax),density=True,color='b',label='Initial');
plt.hist(A3, bins=128,alpha=0.50,range=(rmin,rmax),density=True,color='r',label='Average');
plt.legend(loc='upper left')
plt.ylim(0,ymax)
plt.xlabel('a [AU]');
plt.text(3.55,1.9,'(c)',va='top',ha='right')
plt.ylabel('Number of Asteroids');
```

```
C:\Users\jeffr\anaconda3\lib\site-packages\numpy\lib\histograms.py:905: RuntimeWarning: inval
id value encountered in true_divide
  return n/db/n.sum(), bin_edges
```



Exploration of the 3:1 resonance to long times

We now explore the 3:1 for much longer time. Since this is further than we pushed the benchmarks, let's do at least two different time steps -- I'll pick 1 day and 0.5 days -- so that we can monitor how well they agree. After some experimentation it seems we need to go 0.5 million years to see some interesting behaviour.

In [149…

```
t1=500000*years
%time T1,A1,E1 = resonant_orbit(3.00,t1,e=0.15,θ=0*np.pi,dt=1*days)
%time T2,A2,E2 = resonant_orbit(3.00,t1,e=0.15,θ=0*np.pi,dt=0.5*days)
```

```
Wall time: 53min 6s
Wall time: 1h 45min 27s
```

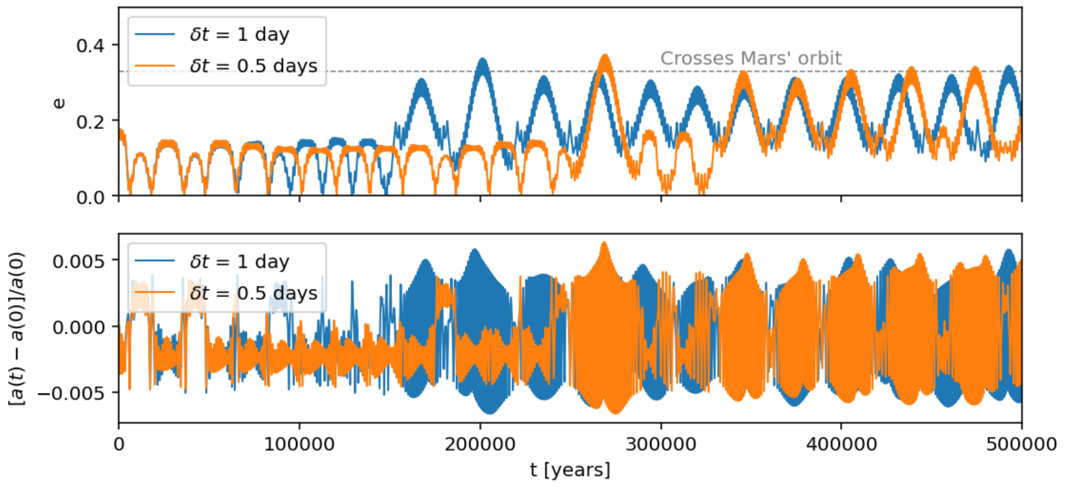
In [189…

```
plt.rcParams["figure.figsize"] = (8.5,4);
fig,axs = plt.subplots(2,1);

# given the absurd number of data points here we won't plot them all. What
# we'll do is only plot them once for each orbit of Jupiter. This can be implemeted
# by skipping a number of steps according to the formula below:
tskip1 = int(jupiter_period/days)
tskip2 = int(jupiter_period/(0.5*days))

# plot the eccentricity
axs[0].plot(T1[::tskip1],E1[::tskip1],lw=1,label="$\delta t$ = 1 day")
axs[0].plot(T2[::tskip2],E2[::tskip2],lw=1,label="$\delta t$ = 0.5 days")
axs[0].set_xlabel('')
axs[0].set_ylabel('e')
axs[0].set_ylim(0,0.5)
axs[0].set_xlim(0,t1)

# given Mars is an important demarcation line, we show the point at which this
# orbit crosses Mars' orbit
axs[0].plot([0,t1],[0.33,0.33],zorder=0,ls='dashed',color='gray',lw=0.75)
axs[0].text(350000,0.34,"Crosses Mars' orbit",color='gray',ha='center',va='bottom')

axs[0].legend()
axs[0].set_xticklabels([])

# make a second plot with the semi-major axis (since only changes <1% or so)
axs[1].plot(T1[::tskip1],1*(A1[::tskip1]-A1[0])/A1[0],lw=1,label="$\delta t$ = 1 day")
axs[1].plot(T2[::tskip2],1*(A2[::tskip2]-A2[0])/A2[0],lw=1,label="$\delta t$ = 0.5 days")
```

```
axs[1].set_xlabel('t [years]')
axs[1].set_ylabel('$[a(t) - a(0)]/a(0)$')
axs[1].set_xlim(0,t1)
axs[1].legend()
```

Out[189…

```
<matplotlib.legend.Legend at 0x1aada823cd0>
```



We see evidence for the chaotic behaviour discussed in the references, with the two time steps disagreeing at long times, but both remaining physically reasonable. We also see that over time frames of 100,000 years or so that the eccentricity of the asteroid's orbit varies erratically and can become large enough to have the asteroid's orbit cross Mars' orbit.

To explore the notion of chaos here, let's simulate the same system (here I'll fix a step of 1 day) with slightly different initial eccentricities. I'll take differences of 10^{-4} and 10^{-8} for simplicity and evolve them for 100,000 years.

In [229…

```
t1=100000*years
%time T3,A3,E3 = resonant_orbit(3.00,t1,e=0.15+1e-8,θ=0*np.pi,dt=1*days)
%time T4,A4,E4 = resonant_orbit(3.00,t1,e=0.15+1e-4,θ=0*np.pi,dt=1*days)
```

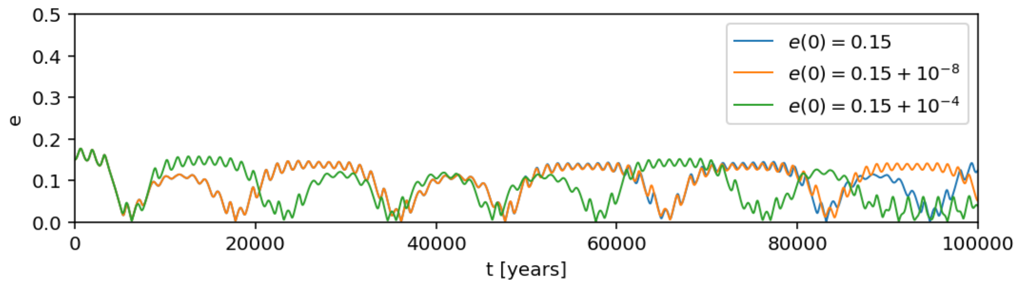
```
Wall time: 10min 39s
Wall time: 10min 28s
```

In [231…

```
plt.rcParams["figure.figsize"] = (8.5,2);
fig,axs = plt.subplots(1,1);
tskip1 = int(jupiter_period/days)
tskip2 = int(jupiter_period/(0.5*days))

plt.plot(T1[::tskip1],E1[::tskip1],lw=1,label="$e(0)=0.15$")
plt.plot(T3[::tskip1],E3[::tskip1],lw=1,label="$e(0)=0.15+10^{-8}$")
plt.plot(T4[::tskip1],E4[::tskip1],lw=1,label="$e(0)=0.15+10^{-4}$")
plt.xlabel('t [years]')
plt.ylabel('e')
plt.ylim(0,0.5)
plt.xlim(0,t1)
plt.legend()
```

Out[231…

```
<matplotlib.legend.Legend at 0x1ab493a0a30>
```



We see that even with a tiny $\Delta e = 10^{-8}$ at the start, over 100,000 years we qualitative differences in the orbit. This is good evidence that in this regime the motion is chaotic. Note that 100,000 years is only 10^4 of Jupiter's periods so any *natural* difference would not be expected yet from the smaller of the Δe.

Benchmarks

We're going to need to track the these orbits for hundreds or thousands of periods of Jupiter's orbit to get an idea what's going on. Therefore we need to be sure our time step is sufficient. To do so we'll first consider a single 2:1 resonant orbit to get an idea of the needed precision.

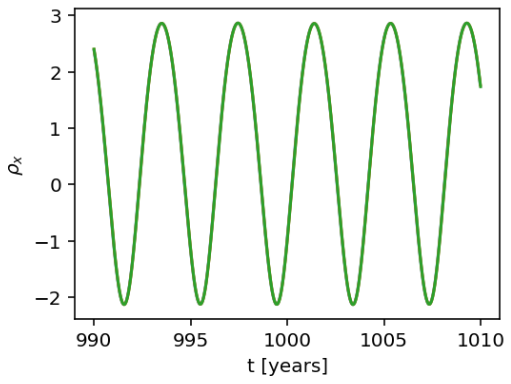
In [ ]:

```
%%time
plt.rcParams["figure.figsize"] = (4,3);

# three different time steps
dt1,dt2,dt3 = 10*days, 5*days, 1*days
t1 = 10000*years

# start a orbit near the 2:1 resonance
u0 = elliptical_orbit_alt(2.5,0.15,θ=0)

T1,U1 = evolve(sun_jupiter,0.0,t1,int(t1/dt1),u0,step=rk4)
T2,U2 = evolve(sun_jupiter,0.0,t1,int(t1/dt2),u0,step=rk4)
T3,U3 = evolve(sun_jupiter,0.0,t1,int(t1/dt3),u0,step=rk4)
```

Let's look at a short window (roughly T_J) at the 1000 year mark.

In [118…

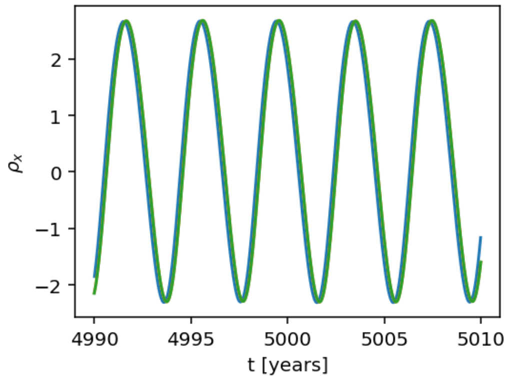
```
region1 = np.abs(T1-1000*years)<10*years
region2 = np.abs(T2-1000*years)<10*years
region3 = np.abs(T3-1000*years)<10*years
plt.plot(T1[region1]/years,U1[region1,0])
plt.plot(T2[region2]/years,U2[region2,0])
plt.plot(T3[region3]/years,U3[region3,0])

plt.xlabel('t [years]');
plt.ylabel('$ρ_x$');
```



We see that the solutions all match quite well. We can then move out to 5000 years.

In [120…

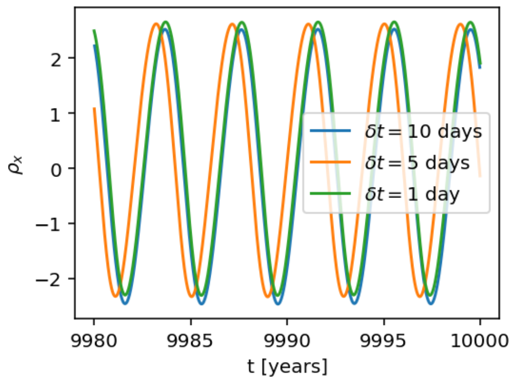
```
region1 = np.abs(T1-5000*years)<10*years
region2 = np.abs(T2-5000*years)<10*years
region3 = np.abs(T3-5000*years)<10*years
plt.plot(T1[region1]/years,U1[region1,0])
plt.plot(T2[region2]/years,U2[region2,0])
plt.plot(T3[region3]/years,U3[region3,0])

plt.xlabel('t [years]');
plt.ylabel('$ρ_x$');
```



We start to see some differences in the longest time-step (10 days). Finally near the end of the interval we can see differences between all three, though they are not very significant.

In [125…

```
region1 = np.abs(T1-9990*years)<10*years
region2 = np.abs(T2-9990*years)<10*years
region3 = np.abs(T3-9990*years)<10*years
plt.plot(T1[region1]/years,U1[region1,0],label='$\delta t = 10$ days')
plt.plot(T2[region2]/years,U2[region2,0],label='$\delta t = 5$ days')
plt.plot(T3[region3]/years,U3[region3,0],label='$\delta t = 1$ day')

plt.xlabel('t [years]');
plt.ylabel('$ρ_x$');
plt.legend();
```



From this we conclude that for times of order a thousand years 10 days or less is a perfectly fine time step. For closer to 10,000 years we see that we likely need a time step of at most day (likely somewhat less). Given this orbit becomes chaotic at around this time scale there are some fundamental limitations to obtaining further accuracy regardless. We will see this when we examine the 3:1 resonance in detail.