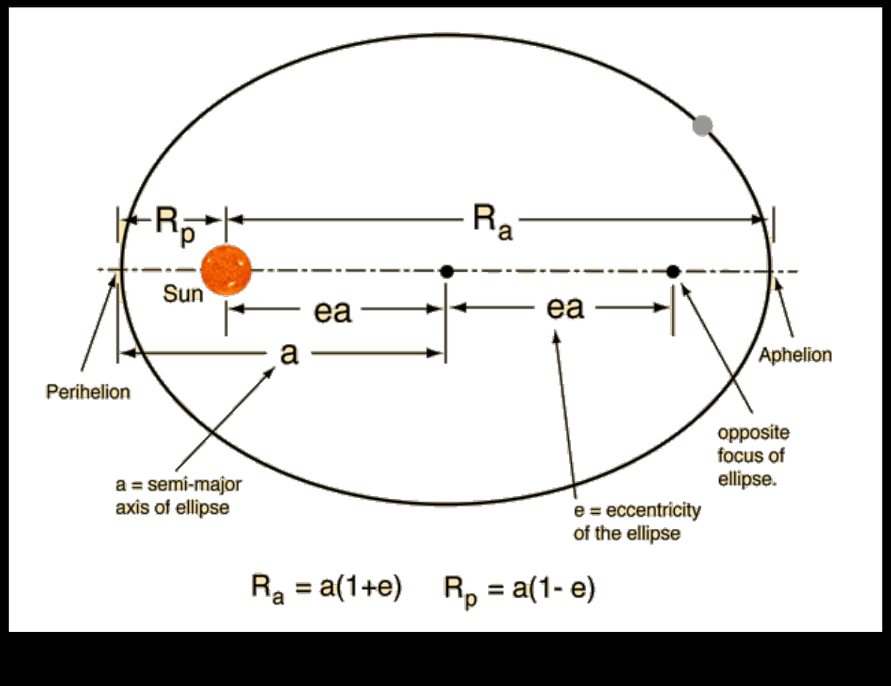
Intro - Day 2

Everything for today is posted under day 2 of:
www.astroblend.com/ba2016

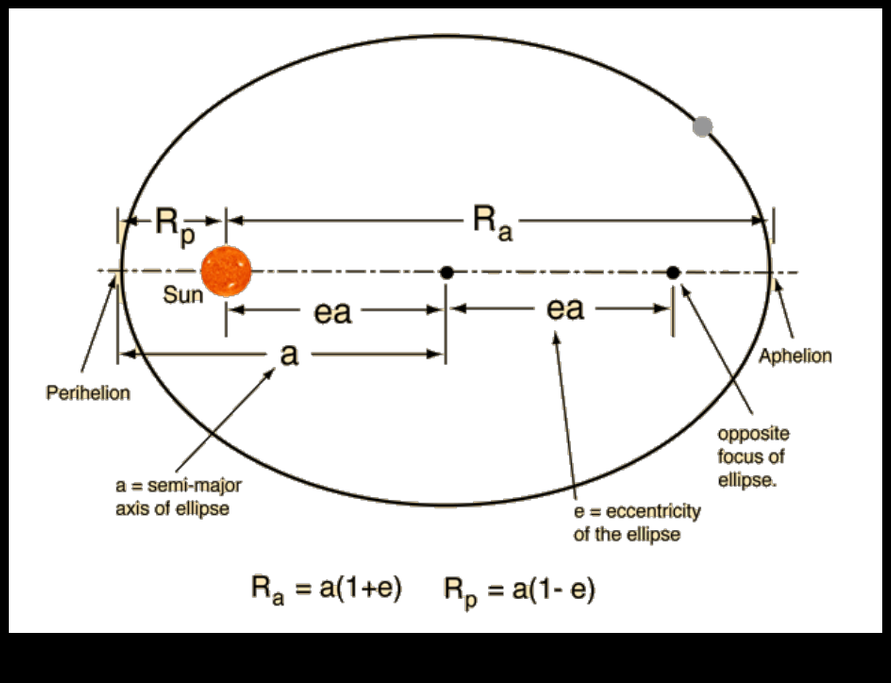


* For the 2-Body problem we compared the analytical and numerical solutions
* Found that $\Delta t << P$ for accurate solutions
* How is this applicable to other simulations? (t_{dyn}, t_{cross})

Intro - Day 2

Everything for today is posted under day 2 of:

www.astroblend.com/ba2016



* For the 2-Body problem we compared the analytical and numerical solutions
* Found that $\Delta t << P$ for accurate solutions
* How is this applicable to other simulations? (t_{dyn}, t_{cross})

The crossing time and virial equilibrium

- The system "quickly" reaches a "steady state"
- The steady state is in "virial equilbrium", when the virial equation is approximately satisfied:

$$2T + V = 0$$

where

$$T = \frac{1}{2}\sum_{i=1}^{N} m_i \mathbf{v}_i^2 \quad \text{(Kinetic Energy)}$$

$$V = -\frac{G}{2}\sum_{i=1}^{N}\sum_{j=1,\neq i}^{N} \frac{m_j m_i}{|\mathbf{r}_i - \mathbf{r}_j|} \quad \text{(Potential Energy)}$$

Mass, length and time scales

- Total mass

$$M = \sum_{i=1}^{N} m_i$$

- Characterise system size by "virial radius" R defined by

$$V = -\frac{GM^2}{2R}, \quad \text{where } M \text{ is total mass}$$

- Characterise speeds by (mass weighted) mean square speed

$$v^2 = \frac{2T}{M}$$

- Define time scale

$$t_{cr} = \frac{2R}{v} \quad (\text{"Crossing time"})$$

Mass, length and time scales

- Total mass

$$M = \sum_{i=1}^{N} m_i$$

- Characterise system size by "virial radius" R defined by

$$V = -\frac{GM^2}{2R}, \quad \text{where } M \text{ is total mass}$$

- Characterise speeds by (mass weighted) mean square speed

$$v^2 = \frac{2T}{M}$$

- Define time scale

$$t_{cr} = \frac{2R}{v} \quad (\text{"Crossing time"})$$

$\Delta t << t_{cr}$ comprable to $\Delta t << P$

Significance of the crossing time

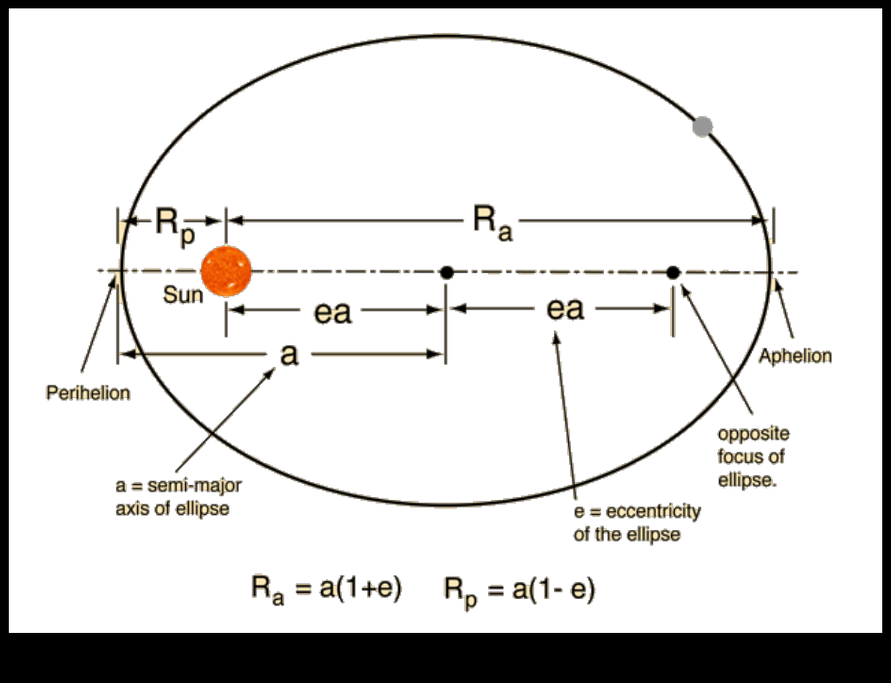
- Time scale of cold collapse
- Time scale of approach to virial equilibrium
- Time scale of orbital motions in virial equilibrium

For hydrodynamical simulations where particles interact with gas properties (heating for example) and gravity the condition is something like: $\Delta t << t_{cr}$ with $t_{cr} \sim R/c_s$

Intro - Day 2

Everything for today is posted under day 2 of:

www.astroblend.com/ba2016

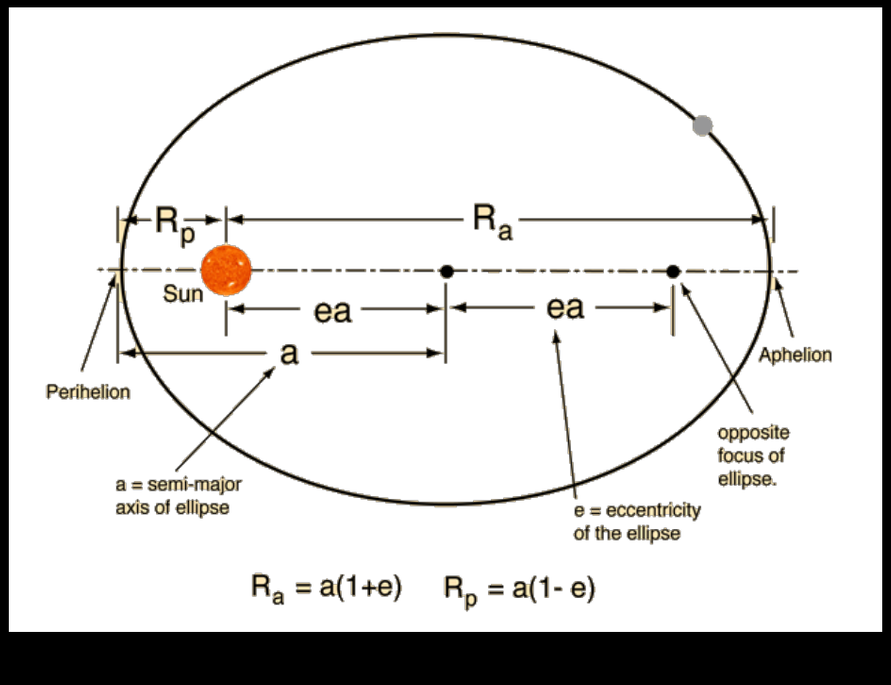


* For the 2-Body problem we compared the analytical and numerical solutions
* Found that $\Delta t << P$ for accurate solutions
* How is this applicable to other simulations? (t_{dyn}, t_{cross})
* Methods to “test” the accuracy of our simulations? (Conservation of energy and momentum)

Intro - Day 2

Everything for today is posted under day 2 of:

www.astroblend.com/ba2016



* For the 2-Body problem we compared the analytical and numerical solutions
* Found that $\Delta t << P$ for accurate solutions
* How is this applicable to other simulations? (t_{dyn}, t_{cross})
* Methods to “test” the accuracy of our simulations? **(Conservation of energy and momentum)**

Calculate Energy & Angular Momentum for Euler's Method

```
# force/mass for particle mi
def calcAcc(mj, ri, rj):
    mag_r = np.sqrt( (ri-rj).dot(ri-rj) )
    return -G*mj*(ri - rj)/mag_r**3.0

# energy
def calcE(mi, mj, ri, rj, vi, vj):
    # energy = 1/2*m1*v1**2 + 1/2*m2*v2**2 - G*m1*m2/|r1 - r2|

# angular momentum
def calcL(mi, mj, ri, rj, vi, vj):
    # ang. mom. = m1*r1 X v1 + m2*r2 X v2


# initial x/y/z coords of masses and their velocities...
r_eu = [np.array([ [0., 0., 0.], [rp, 0., 0.] ])]
v_eu = [np.array([ [0., 0., 0.], [0., vp, 0.] ])]


n_eu = 10000 # number of steps over which to do our calculation
dt = 1e6 # seconds per step

# time array for Euler integration
t_eu = np.linspace(0, dt*n_eu, n_eu)

# energy array
e_eu = []
# angular momentum array
l_eu = []

# loop and calculate
for n in range(1,n_eu):
    # v1_new = (acceleration from mass 2)*dt + v1_old
    # v2_new = (acceleration from mass 1)*dt + v2_old
    # r1_new = 1/2*(acceleration from mass 2)*dt*dt + v1_old*dt + r1_old
    # r2_new = 1/2*(acceleration from mass 1)*dt*dt + v2_old*dt + r2_old
    # v_eu.append( np.array( [v1_new, v2_new] ) )
    # r_eu.append( np.array( [r1_new, r2_new] ) )
    #e_eu.append( calcE(m1, m2, r1, r2, v1, v2) )# note, this is 1 step behind
    #l_eu.append( calcL(m1, m2, r1, r2, v1, v2) )# note, this is 1 step behind

# for plotting, just get x/y coords for m2, we assume
# m1 is fixed at (0,0,0)
x_eu = []
y_eu = []
for n in range(0, n_eu):
    x_eu.append( (r_eu[n][1,0] - r_eu[n][0,0])/AUinCM )
    y_eu.append( (r_eu[n][1,1] - r_eu[n][0,1])/AUinCM )
```

Calculate Energy & Angular Momentum for Euler's Method

```
# plot x/y coords
fig, ax = plt.subplots(1, figsize = (10, 10))
fig.suptitle('Coordinates Plot')

ax.plot(x_an, y_an, linewidth=5)
ax.plot(x_eu, y_eu, linewidth=3)
ax.plot(0.0, 0.0, 'kx')
ax.set_xlabel('x in AU')
ax.set_ylabel('y in AU')
#ax.set_xlim(-3, 3)
#ax.set_ylim(-3, 3)


fig_E, ax_e = plt.subplots(1, figsize = (10, 10))
fig_E.suptitle('Energy Plot')

# E calc is 1 step behind so [1:]
ax_e.plot(t_eu[1:], np.repeat(e_an/e_eu[0], len(t_eu[1:])), linewidth=5)
ax_e.plot(t_eu[1:], e_eu/e_eu[0], linewidth=3)
ax_e.set_xlabel('Time in sec')
ax_e.set_ylabel('Energy(t)/Energy(t=0)')


fig_L, ax_l = plt.subplots(1, figsize = (10, 10))
fig_L.suptitle('Angular Momentum Plot')

# L calc is 1 step behind so [1:]
ax_l.plot(t_eu[1:], np.repeat(l_an/l_eu[0], len(t_eu[1:])), linewidth=5)
ax_l.plot(t_eu[1:], l_eu/l_eu[0], linewidth=3)
ax_l.set_xlabel('Time in sec')
ax_l.set_ylabel('Angular Momentum(t)/Angular Momentum(t=0)')


plt.show()
```