Gravitational *N*-body dynamics

with REBOUND examples

刘尚飞

中山大学物理与天文学院

2024-07-24

Outlines

Introduction to *N*-body problems

Methods of solving *N*-body problems

MISC: softening, collisions, etc.

Addtional forces/effects

REBOUND

Gravity is univeral

Momentum equation（Newton's second law $\vec{F} = m\vec{a}$ ）

$$\rho \frac{\partial \vec{\mathbf{v}}}{\partial t} + \rho\, \vec{\mathbf{v}} \cdot \nabla\, \vec{\mathbf{v}} = -\nabla P + \rho \vec{\mathbf{g}}$$

The knowledge of gravity field in fluid simulations is usually not easy to obtain:

Poisson's equation

$$\nabla \cdot \vec{g} = -4\pi G\rho$$

But in particle-based hydrodynamics (SPH), the gravity is the same as in N-body simulations.

$$F = \mathrm{G}\frac{m_1 m_2}{r^2}$$

艺术想象图：牛顿坐在苹果树下领悟了万有引力

Description of gravitational N-body problems

Each member of an aggregate of N **point mass** bodies with masses m_i $(i = 1, \cdots, N)$, experiences an acceleration that arises from the gravitational attraction of ***all the other bodies*** in the system.

$$\frac{\mathrm{d}^2 \vec{r}_i}{\mathrm{d}t^2} = -\sum_{j=1\ ;\ j\neq i}^{N} \frac{\mathrm{G}m_j \left(\vec{r}_i - \vec{r}_j\right)}{\left|\vec{r}_i - \vec{r}_j\right|^3}$$

The problem is then **completed** by specifying the *initial velocities* $\vec{v}_i\ (t = 0)$ and *positions* $\vec{r}_i\ (t = 0)$ for the N particles.

The ODEs describe astronomical phenomena ranging from orbits of planets and moons to the evolution of star clusters to the spiral structure of galaxies.

The *goal* is to capture, ***as accurately as possible***, the long-term behavior of the N-body system.

- **few-body problems**: accuracy
- **large N problems**: efficiency

Solving *N*-body problems

Rewrite the second-order equations as a set of 2*N* coupled first-order ODEs

$$\frac{\mathrm{d}\vec{r}_i}{\mathrm{d}t} = \vec{v}_i$$

$$\frac{\mathrm{d}\vec{v}_i}{\mathrm{d}t} = -\sum_{j=1\ ;\ j\neq i}^{N} \frac{\mathrm{G}m_j\left(\vec{r}_i - \vec{r}_j\right)}{\left|\vec{r}_i - \vec{r}_j\right|^3}$$

In total, we have 6*N* variables to solve (*integrate*). In ***Euler*** method, one can construct a basic finite difference scheme with a constant time interval Δt to proceed the integration

$$\vec{r}_{i,\,t+\Delta t} = \vec{r}_{i,\,t} + \vec{v}_{i,\,t}\Delta t$$

$$\vec{v}_{i,\,t+\Delta t} = \vec{v}_{i,\,t} + \vec{F}_{i,\,t}\Delta t$$

We know Euler integration method is rudimentary, but why we say that?

Conservation laws

$$E_{\text{tot}} = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1\ ;\ j\neq i}^{N}\frac{\mathrm{G}m_i m_j}{r_{ij}} + \sum_{i=1}^{N}\frac{1}{2}m_i v_i^2 + \frac{1}{2}m_{\text{tot}}v_{\text{com}}^2$$

$$\vec{L}_{\text{tot}} = \sum_{i=1}^{N} m_i\left(\vec{r}_i \times \vec{v}_i\right)$$

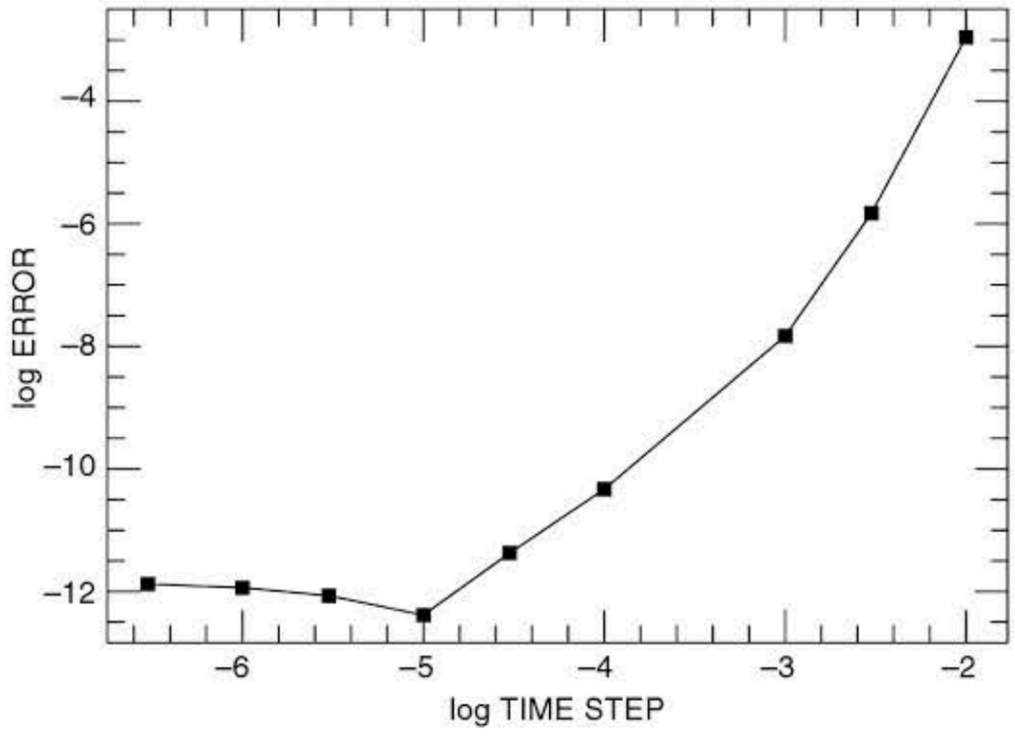
To tell a integration scheme is good or not, one can always check the energy and angular momentum conservation, but not *vice versa*!

In practice, we often use energy conservation

$$\Delta E = \frac{E_{\text{final}} - E_{\text{initial}}}{E_{\text{initial}}}$$

Sun-Earth system in Euler scheme for 100 yrs

$\log \Delta E$ as a function of $\log \Delta t$. So at most after 10^5 year, the energy error builds up to $\sim 10^{-1}$



Why the best accuracy is achieved at a time step $\Delta t \sim 10^{-5}$?

A closer look to the Euler method

Euler method is ***asymmetric*** because the integrated variable solely depends on the initial value.

The ODEs have the form $\dfrac{\mathrm{d}W}{\mathrm{d}t} = g(W)$, so we can write down the Taylor series

$$W^{n+1} = W^n + h\frac{\mathrm{d}W}{\mathrm{d}t} + \frac{h^2}{2!}\frac{\mathrm{d}^2W}{\mathrm{d}t^2} + \cdots$$

The Euler method ignors 2nd-order terms and above. A higher-order integration can be designed to estimate $g(W)$ better over a particular interval h.

A simple improvement is to utilize the slope of $g(t^n, W^n)$ and the Euler method to estimate

$$W\left(t^n + \frac{h}{2}\right) = W_b = W\left(t^n\right) + \frac{h}{2}g(t^n, W^n) \Longrightarrow g_b(t^n + \frac{h}{2}, W_b)\,,$$

which provides a more accurate value for the slope over the interval h.

A classic 4th-order Runge-Kutta method

$$f_a = g(t^n, W^n)$$

$$W_b = W^n + \frac{h}{2} f_a$$

$$f_b = g(t^n + \frac{h}{2}, W_b)$$

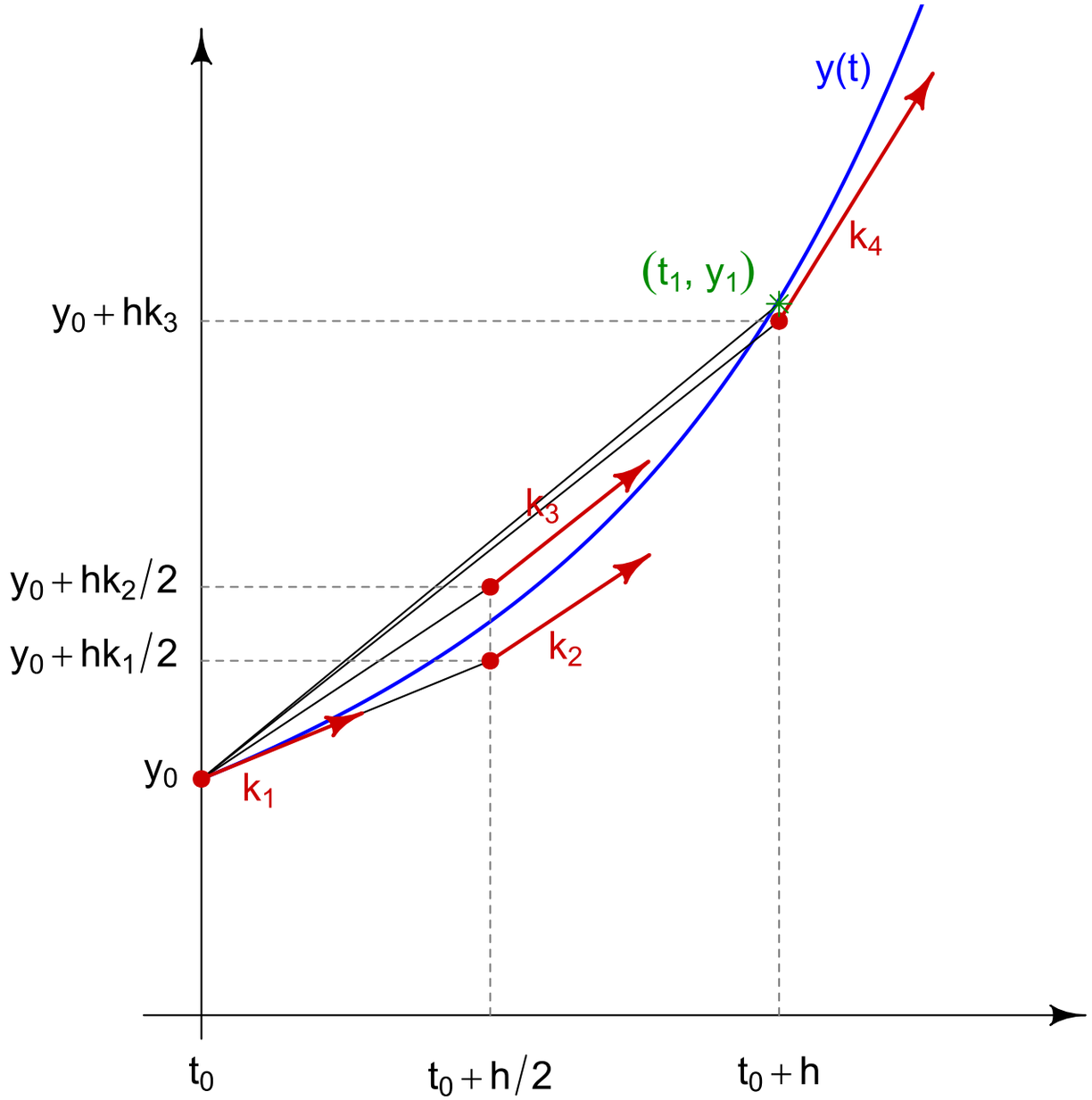
$$W_c = W^n + \frac{h}{2} f_b$$

$$f_c = g(t^n + \frac{h}{2}, W_c)$$

$$W_d = W^n + h f_c$$

$$f_d = g(t^n + h, W_d)$$

$$W^{n+1} = W^n + \frac{1}{6} h f_a + \frac{1}{3} h f_b + \frac{1}{3} h f_c + \frac{1}{6} h f_d$$


y(t)
k_4
(t_1, y_1)
$y_0 + hk_3$
k_3
$y_0 + hk_2/2$
k_2
$y_0 + hk_1/2$
y_0
k_1
t_0
$t_0 + h/2$
$t_0 + h$

Time stepping

In practical, one often needs the ability to adapatively increase or decrease the time step. ***Step doubling*** is a commonly used adaptive step control method. W_1 and W_1' are estimated values at interval h and $2h$, respectively.

$$\delta_e = \left| \frac{W_1 - W_1'}{W_1} \right|$$

If $\delta_e \geq \delta_{\max}$, then adequate convergence has not been achieved. So we need to decrease the time step. And the process is repeated until $\delta_e \leq \delta_{\max}$.

Other high-order *general* methods

- Bulirsch-Stoer integration: a *modified midpoint* method, see e.g., *Numerical Recipies in C*.
- ias15 (Rein & Spiegel 2015): a 15th-order modified Runge-Kutta integrator, default integrator in the rebound code

Symplectic integration

In planetary systems, energy conservation in long-term is a relatively weak constraint. Time reversibility is often more desirable property. The basic idea is to split the total Hamiltonian of the system into two parts.

$$H = H_{\rm kep} + H_{\rm interaction}$$

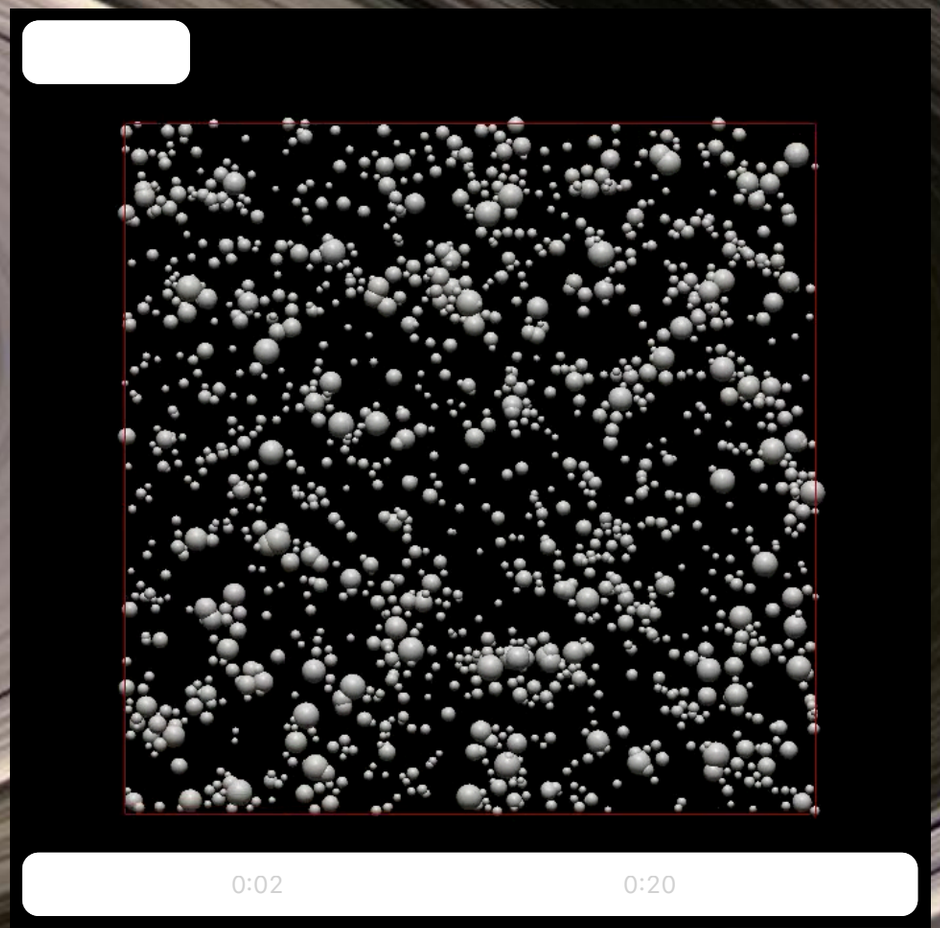
A simple implementaion is the leapfrog method, in which positions of all particles are integrated for half a time-step while keeping the velocities fixed, and then velocities are integrated for half a time-step while positions are fixed.

$$\vec{r}_{n+1} = \vec{r}_n + \vec{v}_{n+1/2}\Delta t$$

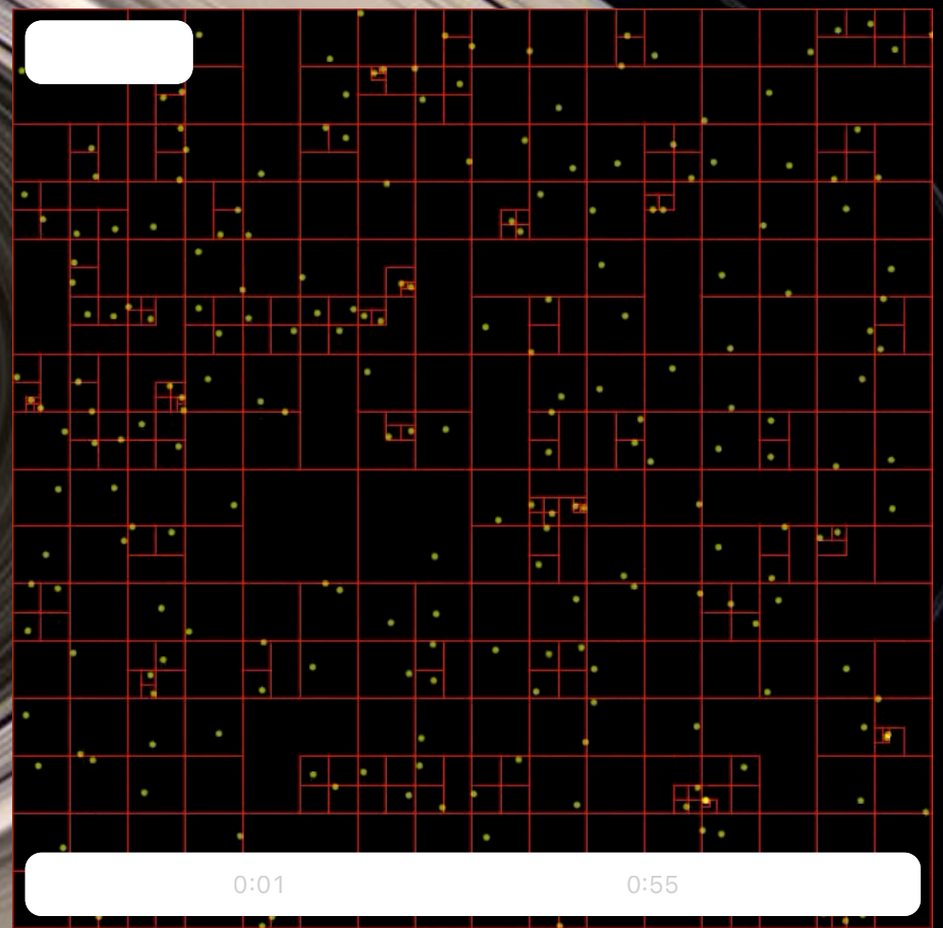
$$\vec{v}_{n+1/2} = \vec{v}_{n-1/2} + \vec{a}_n\Delta t$$

Symplectic integration is ideal for few body problem in planetary systems where planetary interactions are weak. A more advanced symplectic integrator Wisdom-Holman mapping (Wisdom & Holman 1991) is implemented in REBOUND.

N-body for large *N*

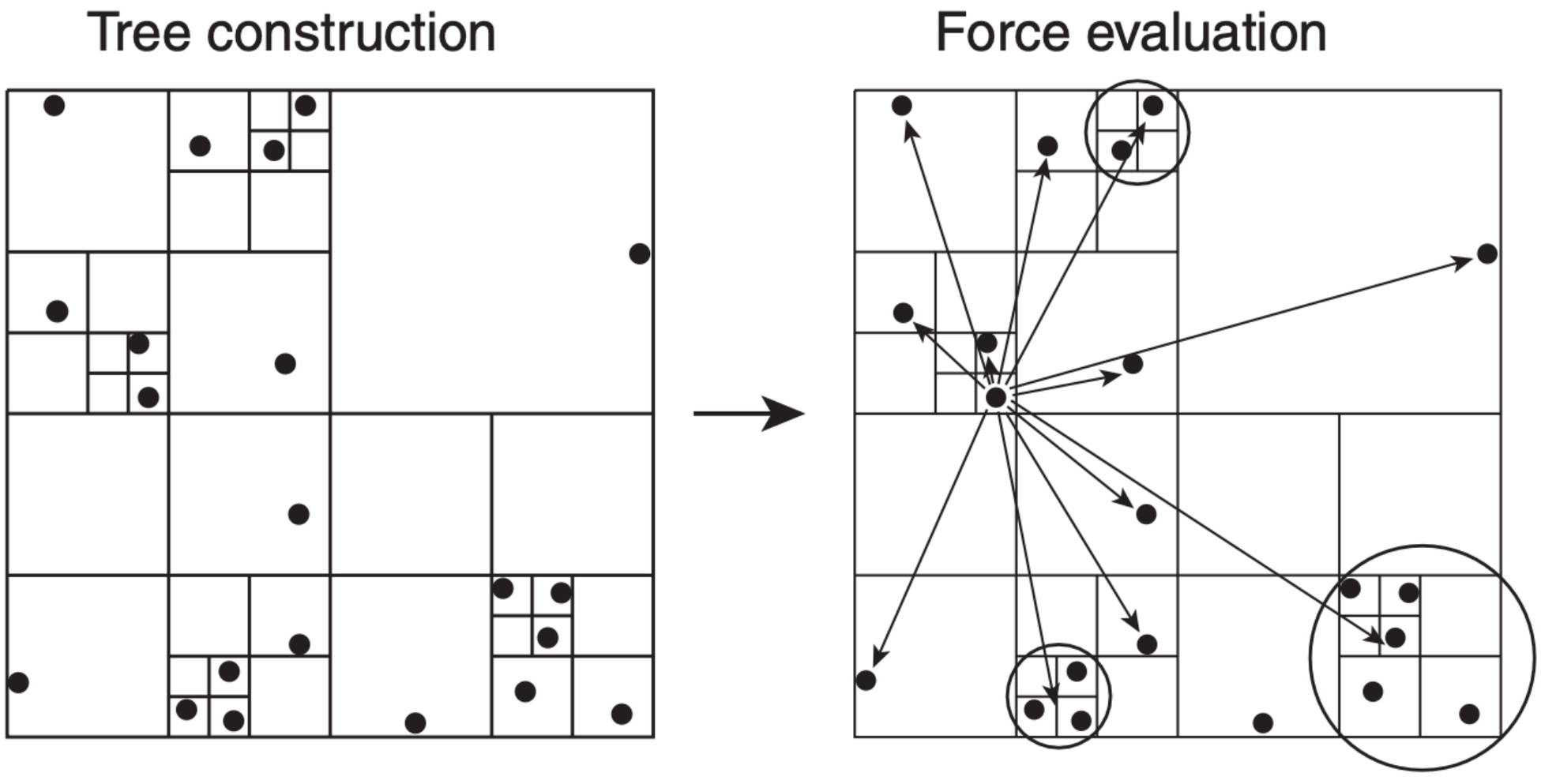


0:02 0:20



0:01 0:55

Tree: approximate method $\sim \mathcal{O}(N \log N)$



Fast Fourier transformation (FFT) $\sim \mathcal{O}(N \log N)$

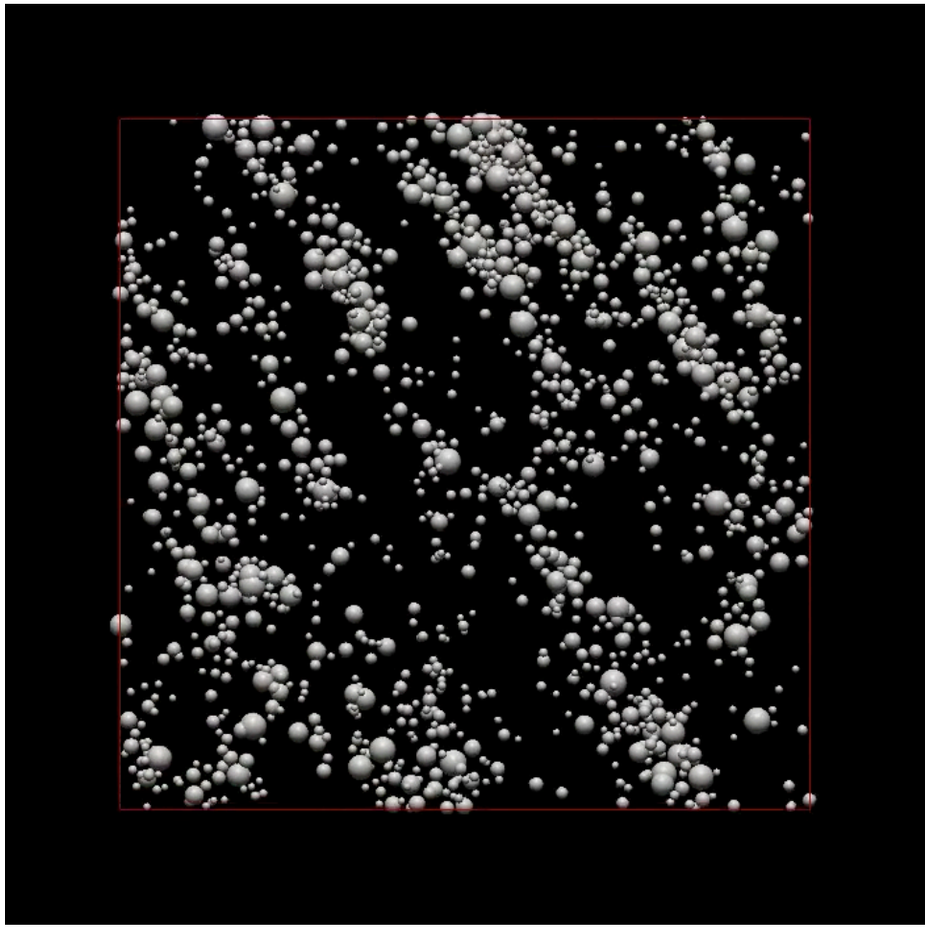
Commonly used in grid-based hydrodynamics.

MISC: softening

$$F = \mathrm{G}\frac{m_1 m_2}{r^2 + \epsilon^2}$$

MISC: collision

A hard-sphere model for colliding pairs



Collision outcomes:

- bouncing: default
- merger: implemented
- fragmentation: not considered in the default version

Additional forces

- disk torques
- gas drag
- magnetic forces, etc.

There is an *additional force* example to show how one can add an additional force into REBOUND, but a more advanced and flexible way to do it is using the REBOUNDx extension (Tamayo et al. 2019)

REBOUND code

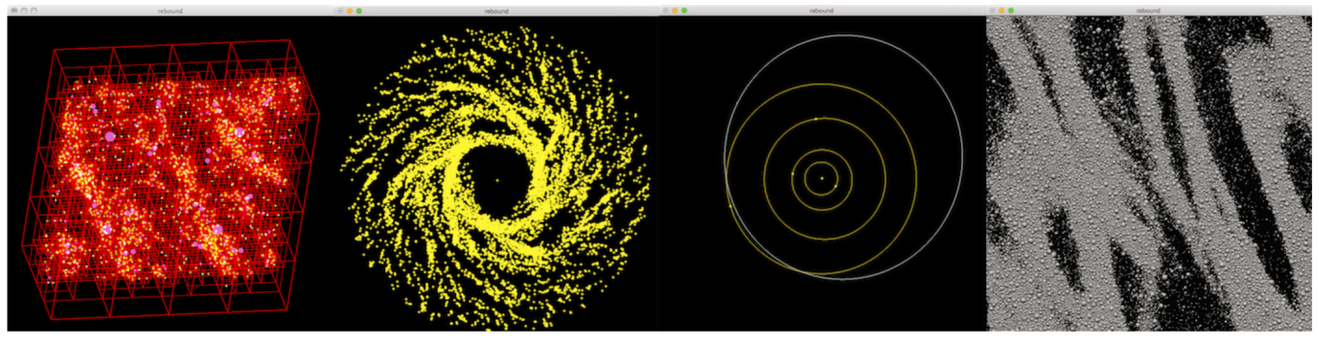
Initially written in C99, now it has a python version

Additional forces/effects can be added using the REBOUNDx extension.

Which version to go?

- Python version: easy to use, can be integrated with other tasks
- C version: easy to parallize with OPENMP and MPI, when you want to modify/add new features

Welcome to REBOUND



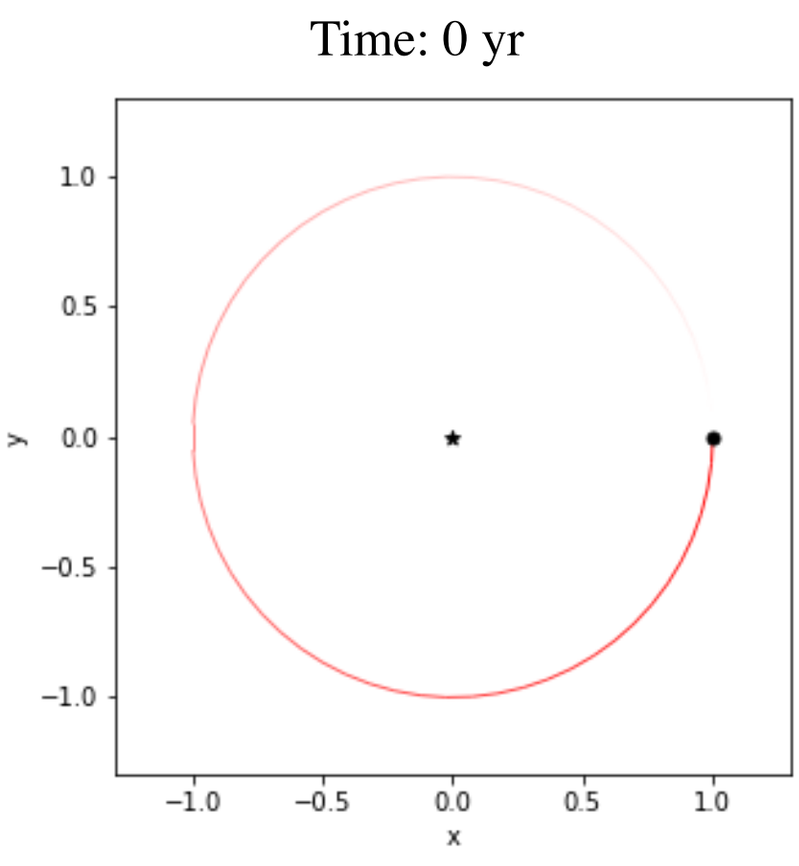
REBOUND is an N-body integrator, i.e. a software package that can integrate the motion of particles under the influence of gravity. The particles can represent stars, planets, moons, ring or dust particles. REBOUND is very flexible and can be customized to accurately and efficiently solve many problems in astrophysics.

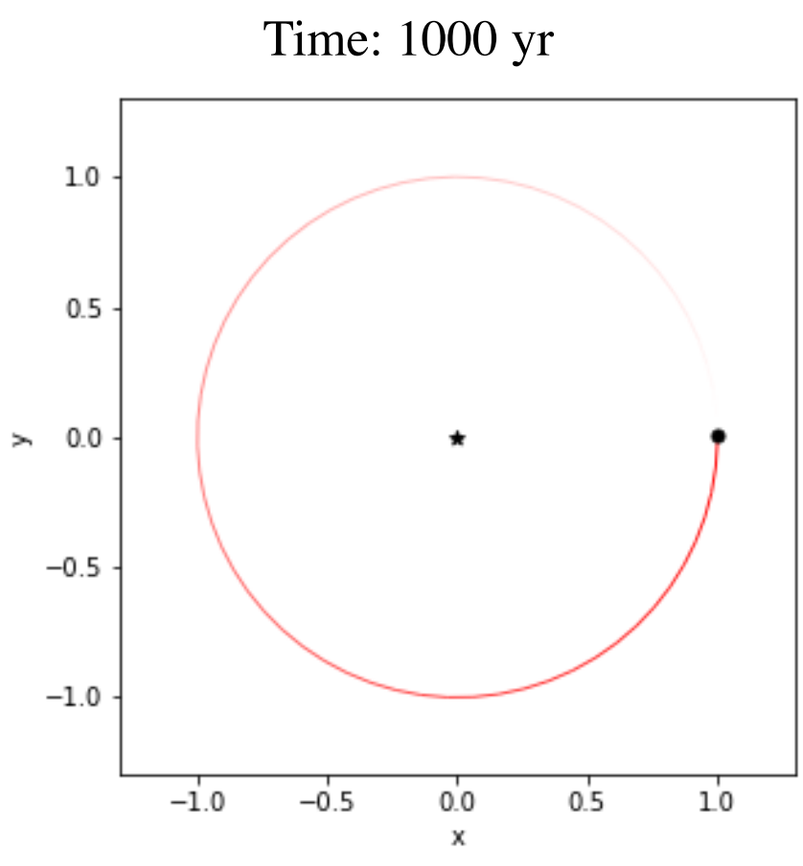
Features

- Symplectic integrators (WHFast, SEI, LEAPFROG, EOS)
- High order symplectic integrators for integrating planetary systems (SABA, WH Kernel methods)
- Hybrid symplectic integrators for planetary dynamics with close encounters (MERCURIUS)
- High accuracy non-symplectic integrator with adaptive time-stepping (IAS15)
- Support for collisional/granular dynamics, various collision detection routines
- The code is written entirely in C, conforms to the ISO standard C99 and can be used as a thread-safe shared library
- Easy-to-use Python module, installation in 3 words: `pip install rebound`
- Extensive set of example problems in both C and Python
- Real-time, 3D OpenGL visualization (C version)
- Parallelized with OpenMP (for shared memory systems)
- Parallelized with MPI is supported for some special use cases only (using an essential tree for gravity and collisions)
- No dependencies on external libraries (use of OpenGL/glfw3 for visualization is optional)
- The code is 100% open-source. All features are inluced in the public repository on github
- No configuration is needed to run any of the example problems. Just type `make && ./rebound` in the problem directory to run them
- Comes with standard ASCII or binary output routines

Visualizing the two-body problem

Plot | Python code





🤔

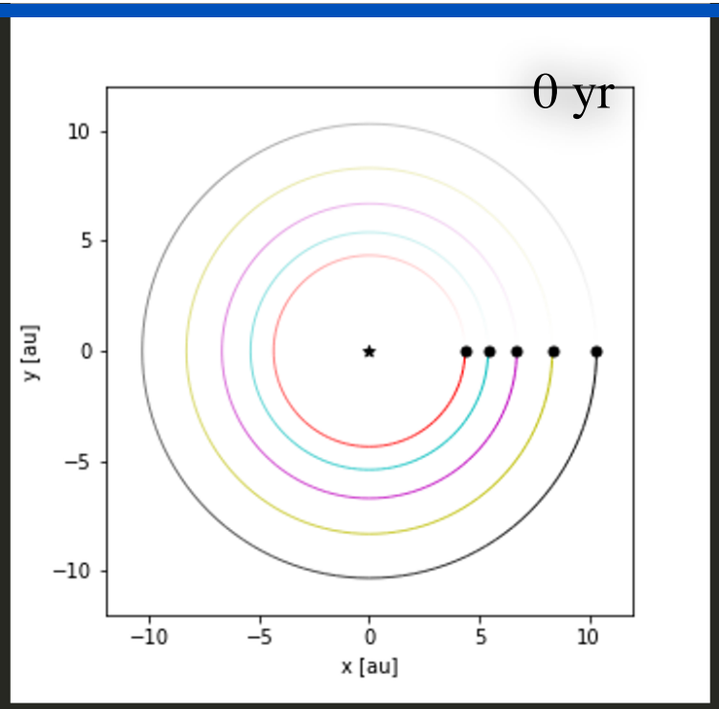
Visualizing the two-body problem

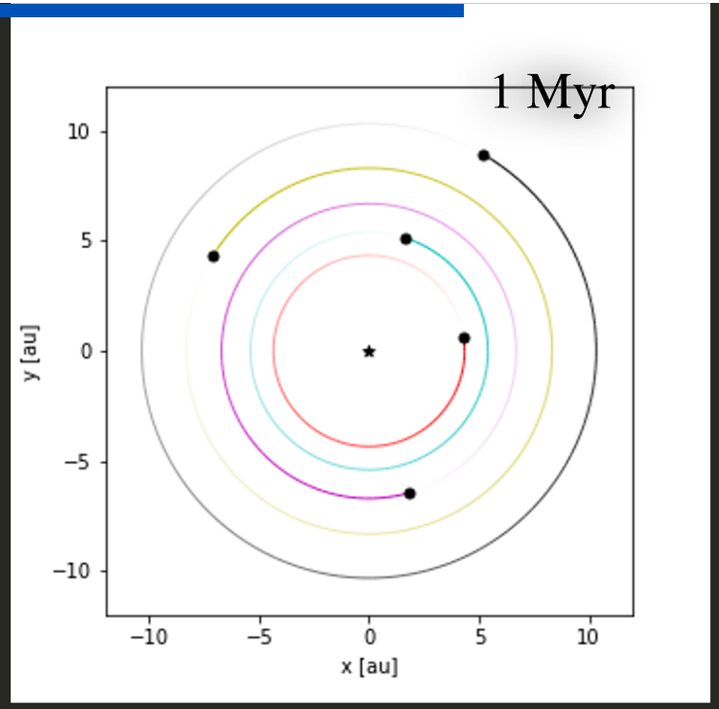
Plot | Python code

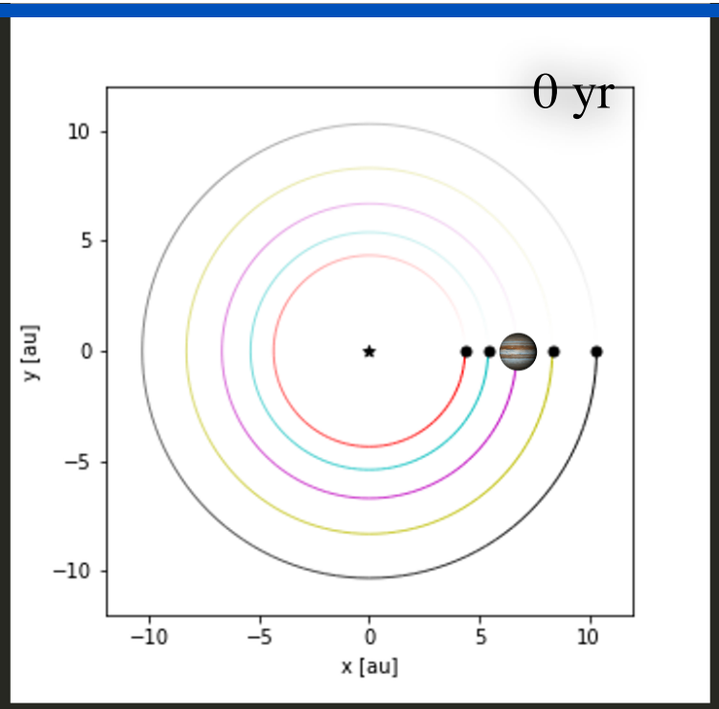
```python
import rebound # Rein & Liu (2012)
import matplotlib.pyplot as plt
import numpy as np

# Initializing a simulation
sim = rebound.Simulation()

# Adding particles into the simulation
sim.add(m=1);
sim.add(m=1e-3/300., a=1) #地球质量约为太阳质量
sim.move_to_com()

# Plot the initial state
fig = rebound.OrbitPlot(sim,color=True)
plt.show()
```

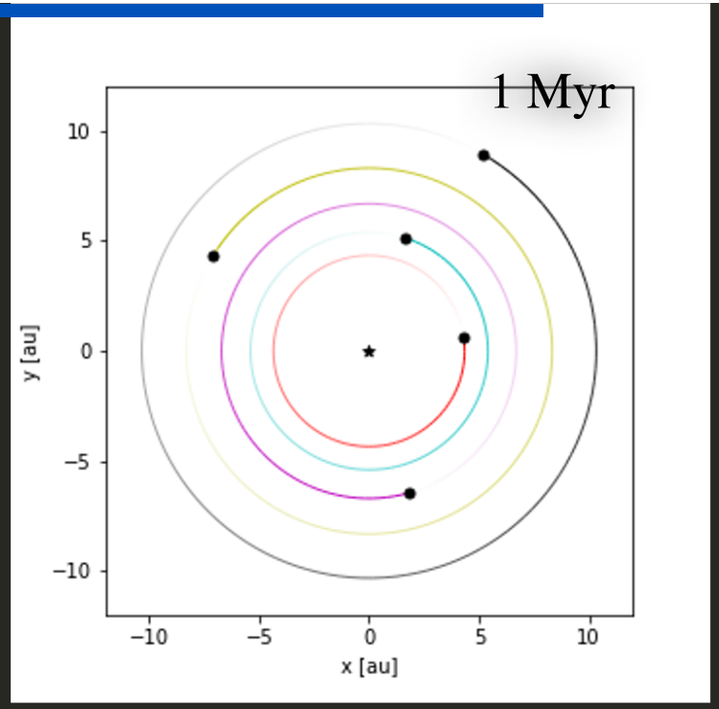
```python
# Integrate for a thousand years
sim.integrate(1000.*2.*np.pi)

# Plot the final state
fig = rebound.OrbitPlot(sim,color=True)
plt.show()
```

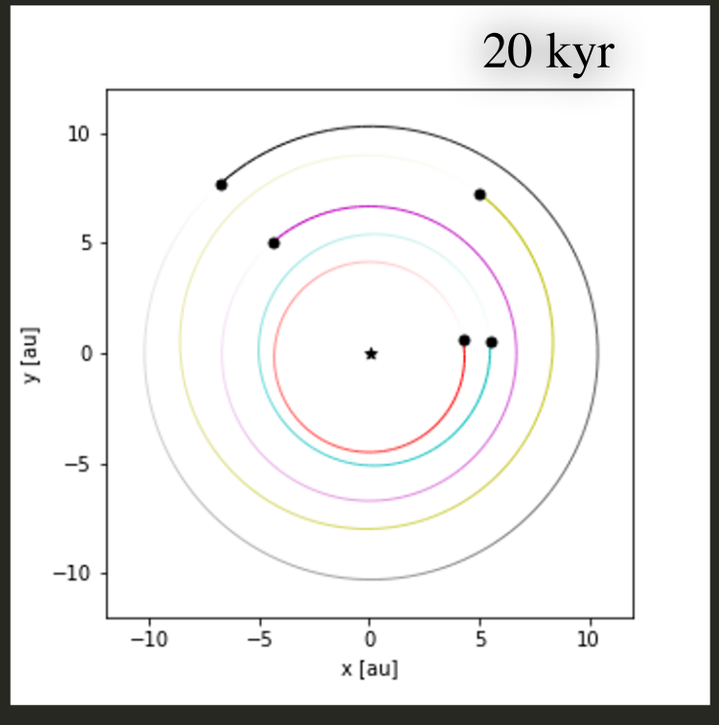
😄 So easy!

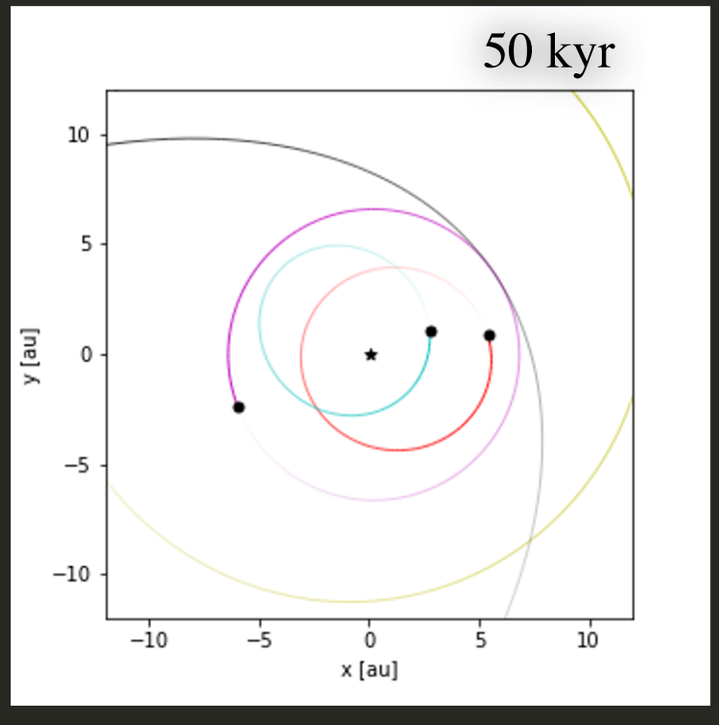

0 yr
y [au]
x [au]
10
5
0
−5
−10
−10
−5
0
5
10


1 Myr
y [au]
x [au]
10
5
0
−5
−10
−10
−5
0
5
10

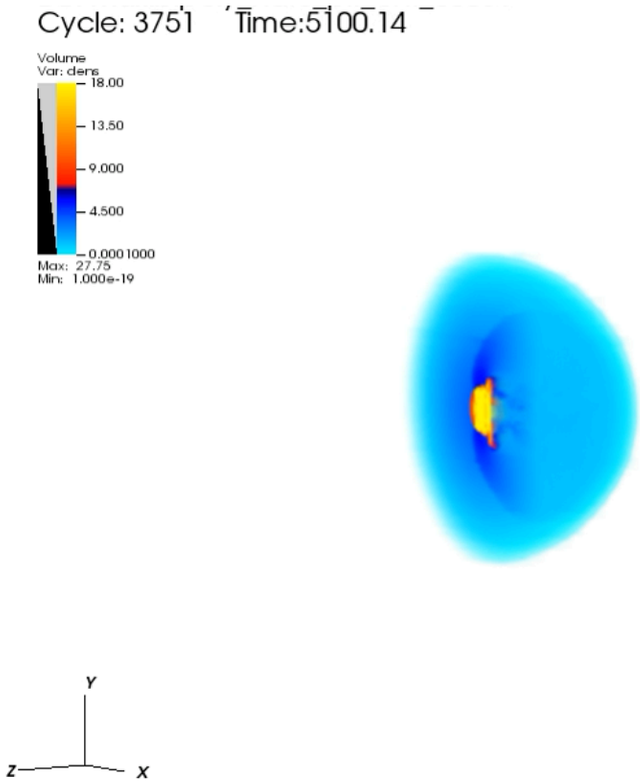

0 yr
y [au]
x [au]


1 Myr
y [au]
x [au]


20 kyr
y [au]
x [au]

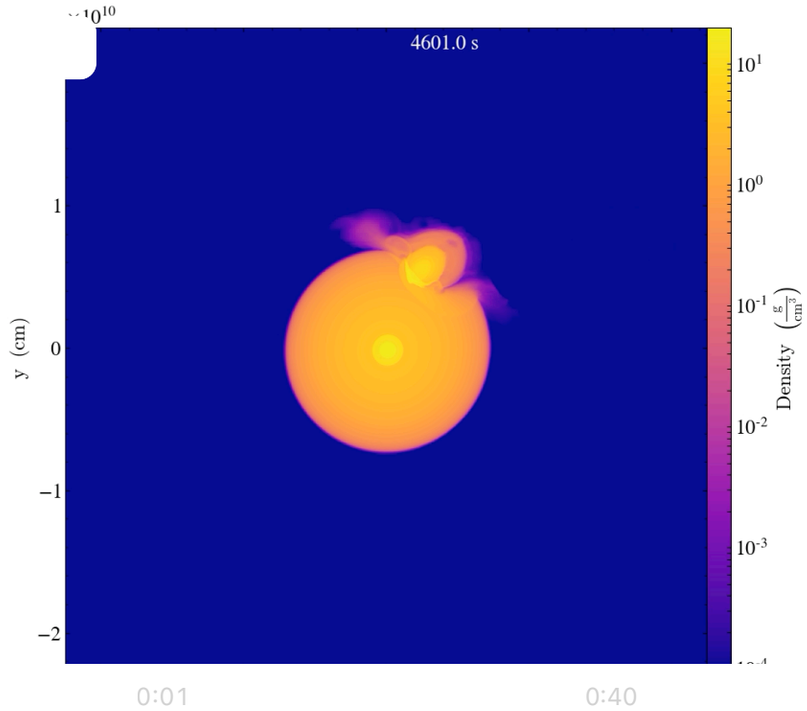

50 kyr
y [au]
x [au]

Collision outcomes

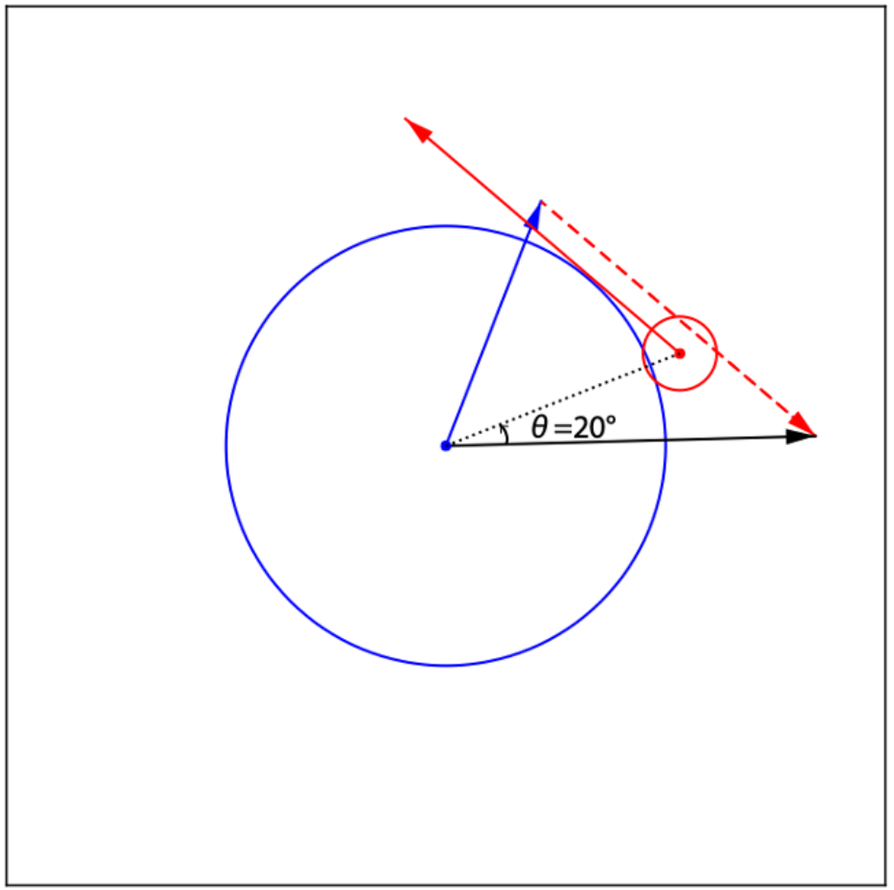


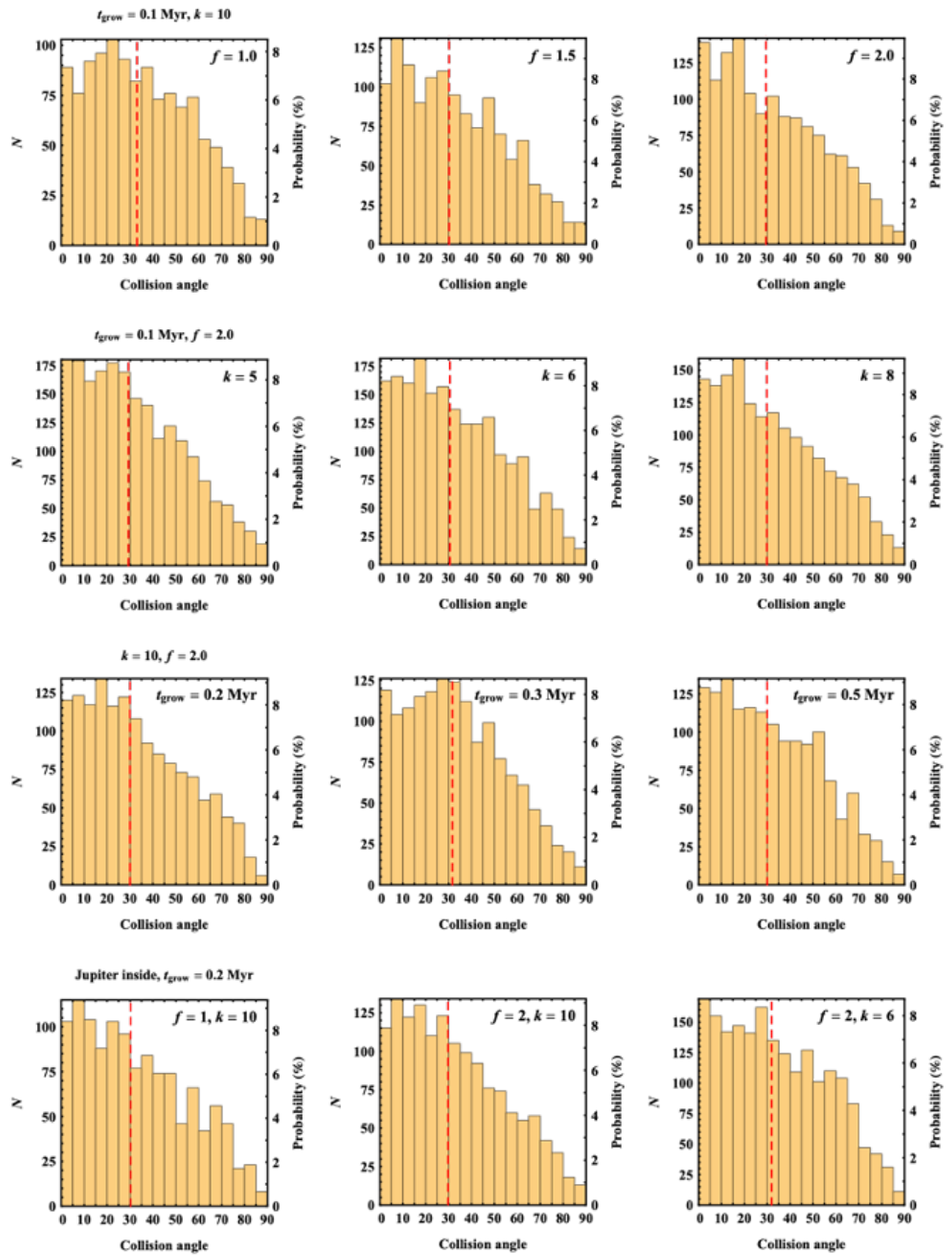
Liu et al. 2019

45度斜碰

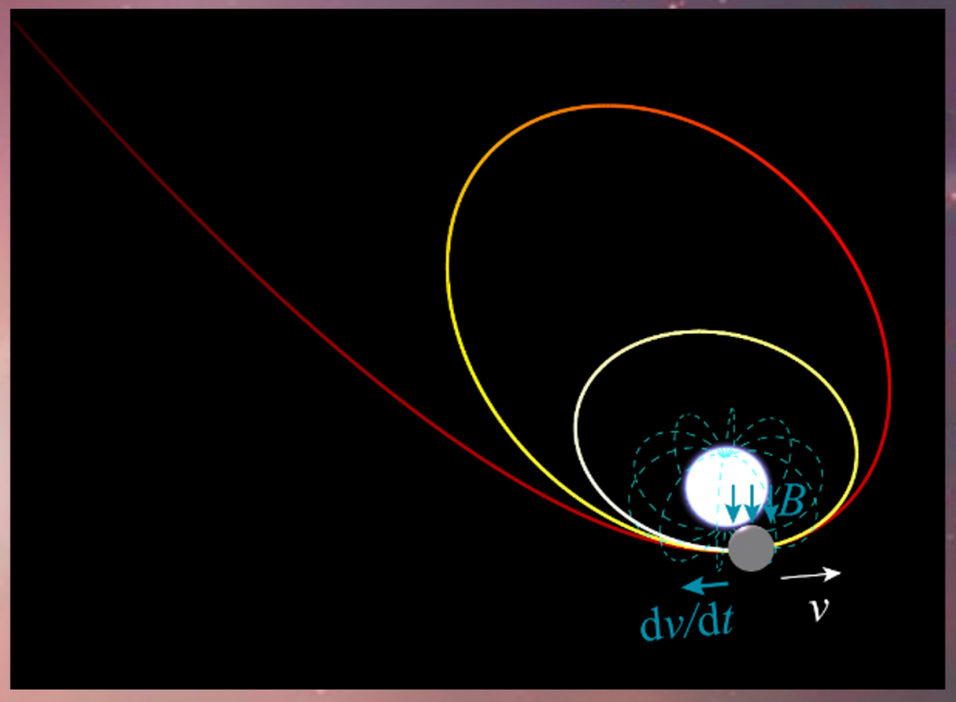


Collision at 20 degrees





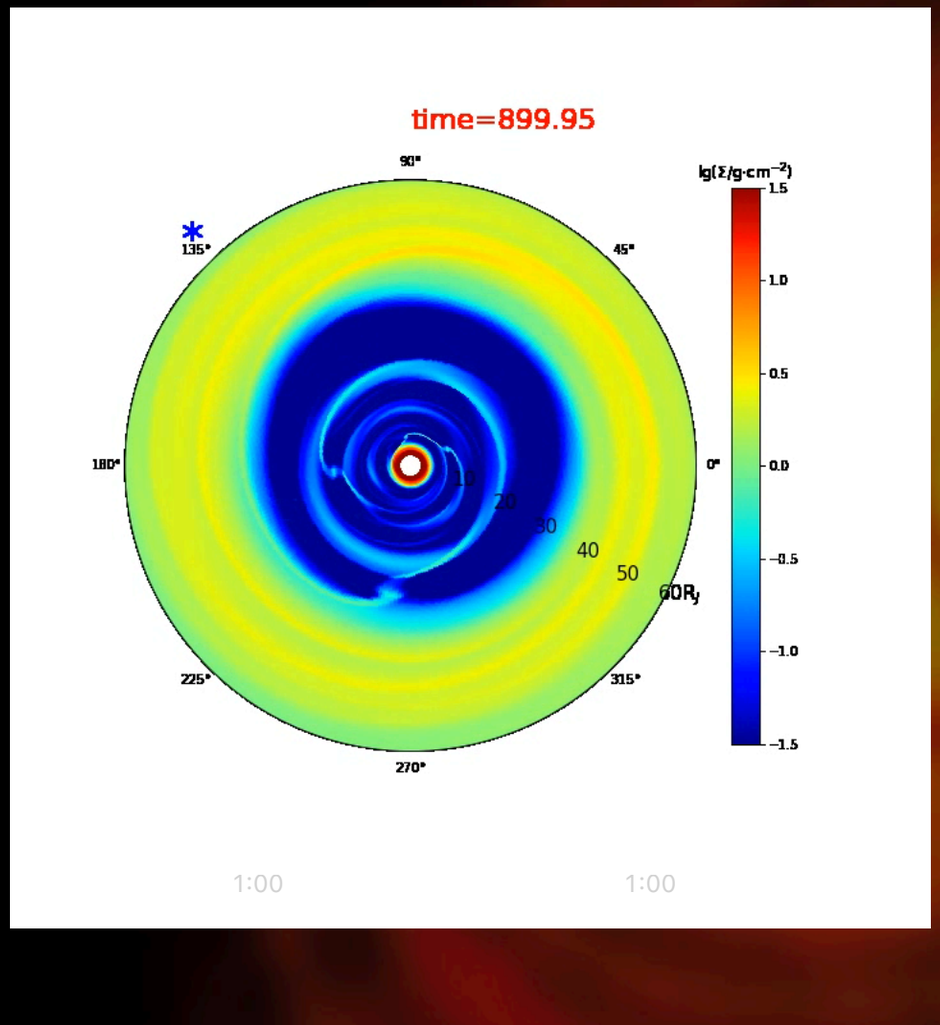
Orbital migration and circularization



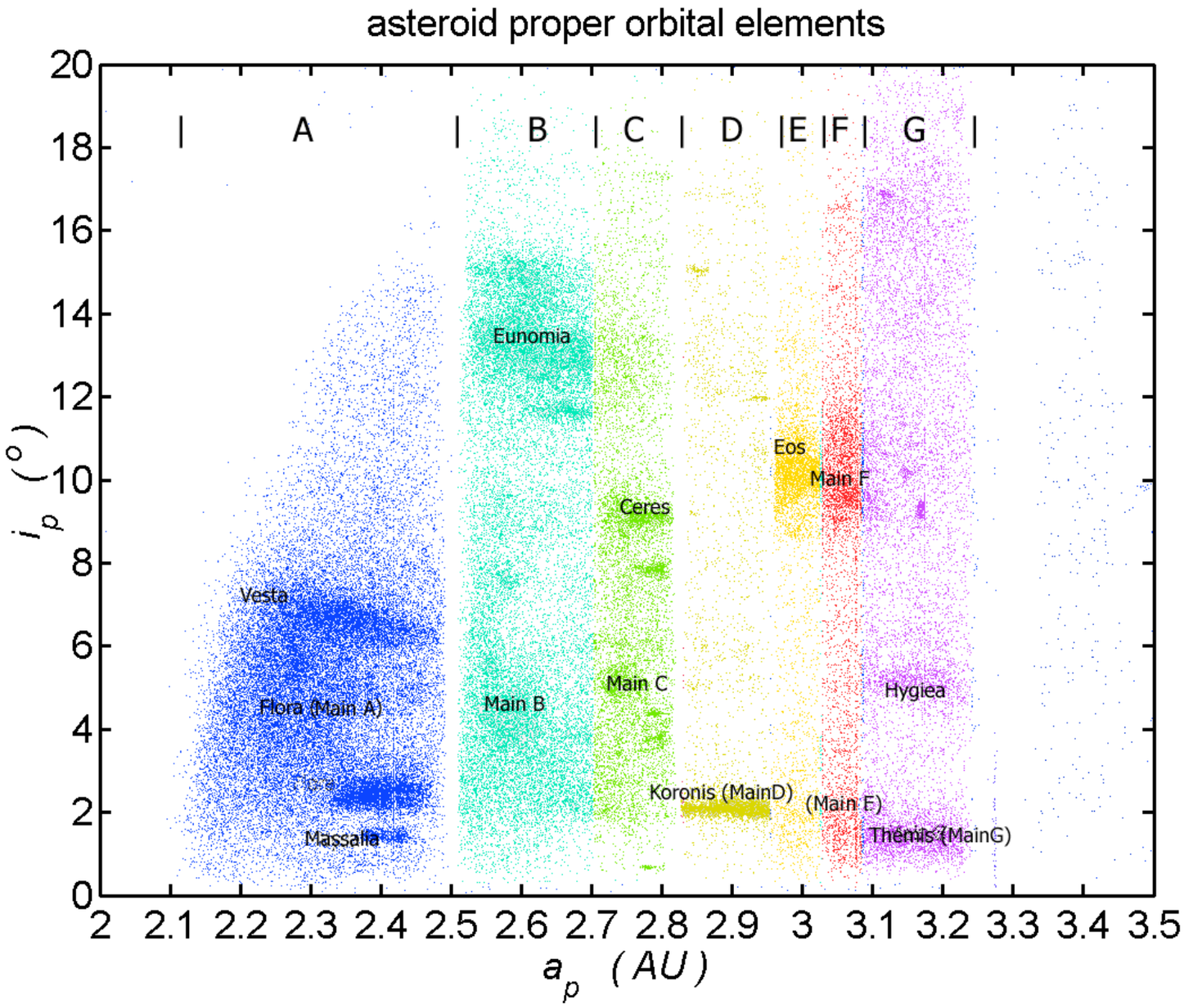
Alfvén wave drag as the main driver

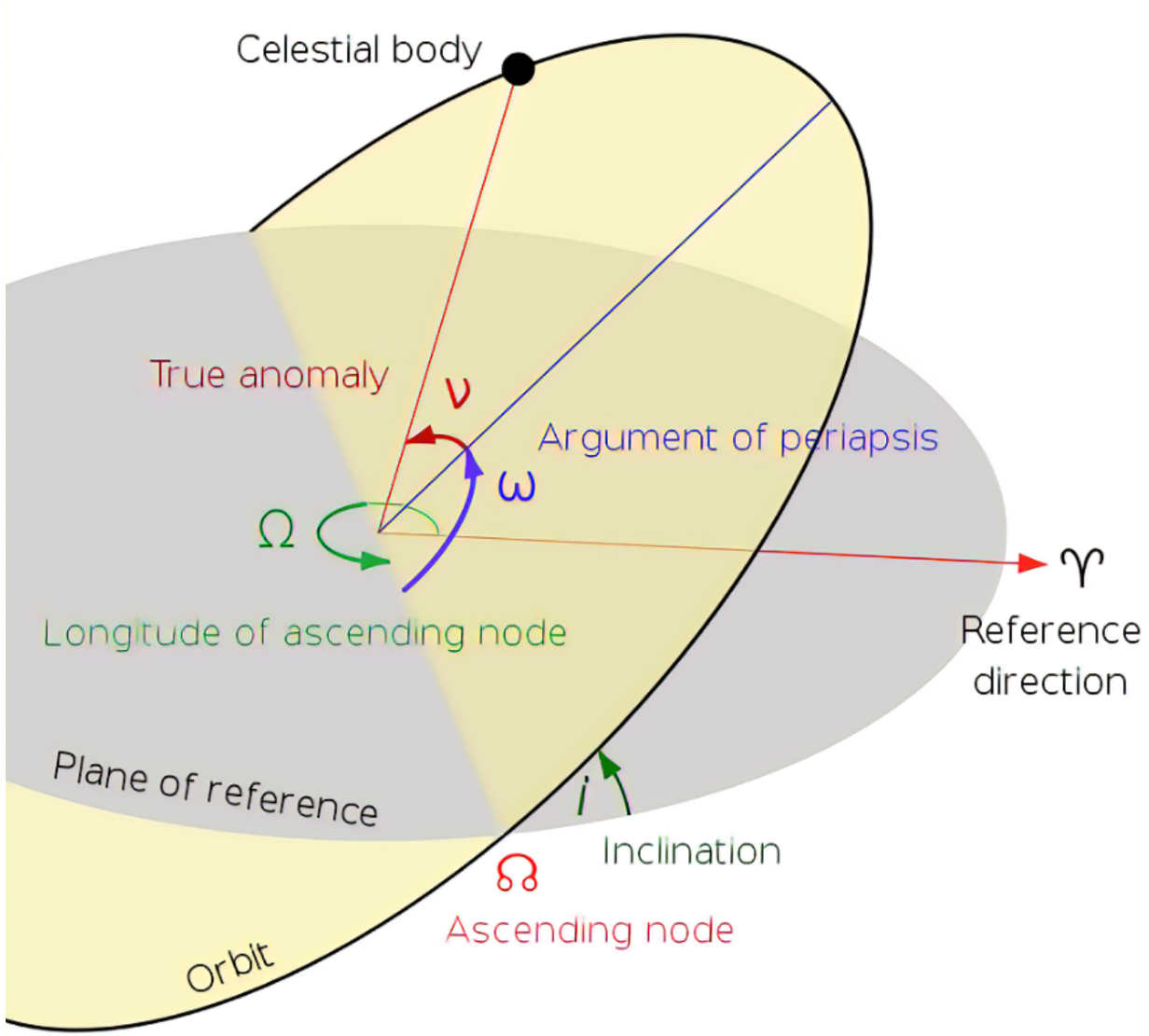
- Tool: REBOUND code (Rein & **Liu** 2012)
- Alfvén wave drag modeled as an additional force
- Parameters:
  - $q \simeq 10^{-3}$ au $= 2R_\odot$
  - $B_* = 100$ T $= 10^6$ G
  - $M_* = 0.6M_\odot$
  - A wide range of semi-major axis a and particle sizes r
- Alfvén wave drag is very efficient!
- Alfvén wave drag modeled as an additional force

Formation of Galilean moon-like systems



- A fourth-order Runge-Kutta integrator has been implemented to calculate orbital evolution
- Incorporation of REBOUND (Rein & Liu 2012) in Athena++ is possible


asteroid proper orbital elements
A
B
C
D
E
F
G
Eunomia
Ceres
Eos
Main F
Vesta
Flora (Main A)
Main B
Main C
Hygiea
Koronis (MainD)
(Main E)
Themis (MainG)
Massalia
i_p (°)
a_p (AU)


Celestial body
True anomaly
ν
Argument of periapsis
ω
Ω
♈
Reference direction
Longitude of ascending node
Plane of reference
i
Inclination
Ascending node
Orbit

References

Numerical Methods in Astrophysics: an introduction, Bodenheimer, Laughlin, Rozyczka, & Yorke 2007

Astrophysics of Planet Formation (2nd ed.), Phil Armitage 2020

Gravitational N-body Simulations: tools and algorithms, Sverre J. AArseth 2003