Computation and astrophysics of the N-body problem

Lecture 3

Outline of Lecture 2

1. Plummer's model
2. Two-body relaxation, dynamical friction
3. Core collapse, long-term evolution
4. Dynamics of binaries

N-body codes

The equations of motion to be integrated are

$$\ddot{\mathbf{r}}_i = -G \sum_{j=1,\neq i}^{N} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3}$$

(Lecture 1).
These can be written in the equivalent form

$$\begin{aligned} \dot{\mathbf{r}}_i &= \mathbf{v}_i \\ \dot{\mathbf{v}}_i &= \mathbf{a}_i = -\sum_{j=1,j\neq i}^{N} G m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} \end{aligned}$$

where $\mathbf{r}_i, \mathbf{v}_i$ are the position and velocity of the ith particle.

The Integration Algorithm

Example: Euler method Algorithm:

$$\begin{aligned} \mathbf{r}_i(t + \Delta t) &= \mathbf{r}_i(t) + \Delta t\, \mathbf{v}_i(t) \\ \mathbf{v}_i(t + \Delta t) &= \mathbf{v}_i(t) + \Delta t\, \mathbf{a}_i(t) \end{aligned}$$

where Δt is the “time step”. In this approximation, the velocity and acceleration of the particle are held constant for the duration of the time step.

Accuracy of the Euler Method

- A more accurate integrator for the position is Taylor's Theorem:

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \Delta t\, \mathbf{v}_i(t) + \frac{1}{2}(\Delta t)^2\, \mathbf{a}_i(t) + \frac{1}{6}(\Delta t)^3\, \mathbf{j}_i(t) + \ldots,$$

  where $\mathbf{j}_i = \dot{\mathbf{a}}_i$, i.e. the derivative of the acceleration, called the "jerk".
- In Euler's algorithm we ignore all except the first two terms. This may be a satisfactory approximation if the series converges sufficiently rapidly.
- The radius of convergence of a Taylor Series is governed by the "distance" to the nearest singularity of the function being expanded. Here, this means the maximum value of Δt for which the function $\mathbf{r}_i(t + \Delta t)$ is well behaved.
- Thus $\Delta t \ll v/r$, where v is the speed, and r is the distance to the nearest neighbour

Time Step Control: Practice

The Taylor series for $\mathbf{r}_i$ is

$$\mathbf{r}_i(t+\Delta t) = \mathbf{r}_i(t) + \Delta t\, \mathbf{v}_i(t) + \frac{1}{2}(\Delta t)^2\, \mathbf{a}_i(t) + \frac{1}{6}(\Delta t)^3\, \mathbf{j}_i(t) + \ldots .$$

This converges rapidly if $\frac{1}{6}(\Delta t)^3\, |\mathbf{j}_i(t)| \ll \frac{1}{2}(\Delta t)^2\, |\mathbf{a}_i(t)|$ (and similar relations between successive pairs of terms), i.e. if

$$\Delta t \ll \frac{|\mathbf{a}_i|}{|\mathbf{j}_i|},$$

This criterion would work well except if, by chance, the denominator were small. Therefore the criterion of choice incoporates several such ratios:

$$\Delta t = \left(\eta \frac{|\mathbf{a}_i|/|\mathbf{j}_i| + |\mathbf{j}_i|/|d\mathbf{j}_i/dt|}{|d^2\mathbf{j}_i/dt^2|/|d\mathbf{j}_i/dt| + |d\mathbf{j}_i/dt|/|\mathbf{j}_i|} \right)^{1/2} .$$

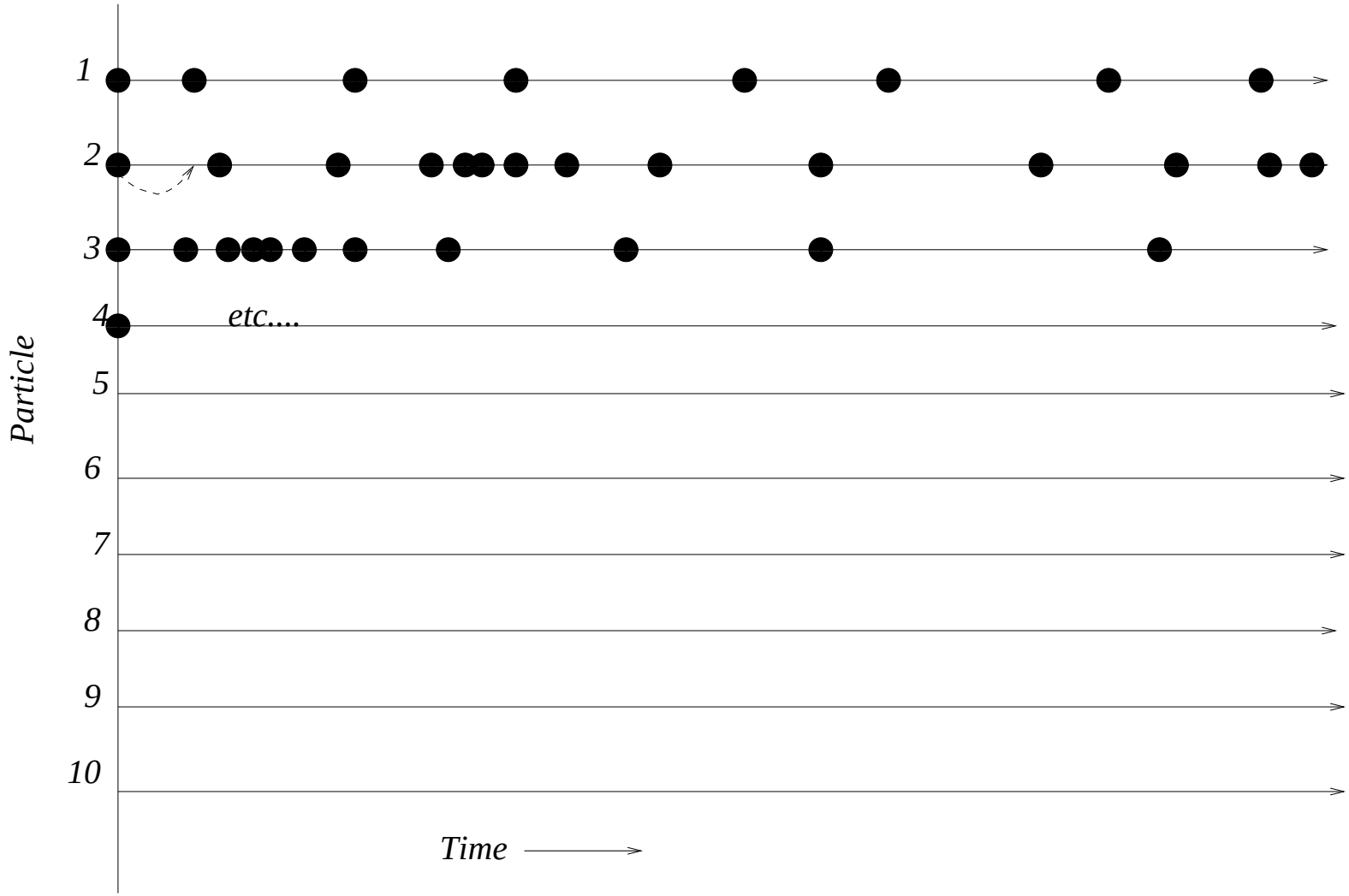
Choices of Time Step

$$\Delta t = \left(\eta \frac{|\mathbf{a}_i|/|\mathbf{j}_i| + |\mathbf{j}_i|/|d\mathbf{j}_i/dt|}{|d^2\mathbf{j}_i/dt^2|/|d\mathbf{j}_i/dt| + |d\mathbf{j}_i/dt|/|\mathbf{j}_i|} \right)^{1/2}.$$

This expression varies with time, and is different for different particles. The following implementations are possible.

- *Fixed* time step (the minimum over all particles for the entire simulation): too short, can't be predicted - impractical
- Variable *shared* time step (the minimum over all particles at the current time) - forces all particles to take same Δt - inefficient.
- Variable *individual* time step - near-optimal, but requires extrapolation.
- Block time steps ($\Delta t = 2^{-k}, k = 0, 1, 2, \ldots$) - shares extrapolation

Individual Variable Time Steps



Extrapolation step: $\mathbf{r}_j(t+\delta t) = \mathbf{r}_j(t) + \delta t \mathbf{v}_j(t)$

The Hermite Integrator

A generalisation of Euler.
First stage:

$$
\begin{aligned}
\text{Euler:}\quad \mathbf{r}_i &:= \mathbf{r}_i + \mathbf{v}_i \Delta t \\
\mathbf{v}_i &:= \mathbf{v}_i + \mathbf{a}_i \Delta t \\
\text{Hermite:}\quad \mathbf{r}_i &:= \mathbf{r}_i + \mathbf{v}_i \Delta t + \frac{1}{2}\mathbf{a}_i \Delta t^2 + \frac{1}{6}\dot{\mathbf{a}}_i \Delta t^3 \\
\mathbf{v}_i &:= \mathbf{v}_i + \mathbf{a}_i \Delta t + \frac{1}{2}\dot{\mathbf{a}}_i \Delta t^2
\end{aligned}
$$

followed by a second stage (corrector) involving values of $\mathbf{a}_i, \dot{\mathbf{a}}_i$ at the *end* of the time step (Hermite only).

The Hermite Integrator: Correction Step

Let $\mathbf{v}, \mathbf{a}, \mathbf{j}$ be velocity, acceleration and jerk at the start of the step, and $\mathbf{v}', \mathbf{a}', \mathbf{j}'$ be the values at the end (computed after the first stage). Then the corrected formulae are

$$\begin{aligned}
\mathbf{r}_i &:= \mathbf{r}_i + \frac{1}{2}(\mathbf{v}_i + \mathbf{v}'_i)\Delta t - \frac{1}{10}(\mathbf{a}'_i - \mathbf{a}_i)\Delta t^2 + \frac{1}{120}(\mathbf{j}_i + \mathbf{j}'_i)\Delta t^3 \\
\mathbf{v}_i &:= \mathbf{v}_i + \frac{1}{2}(\mathbf{a}_i + \mathbf{a}'_i)\Delta t - \frac{1}{12}(\mathbf{j}'_i - \mathbf{j}_i)\Delta t^2.
\end{aligned}$$

Computation of the Jerk

The acceleration is

$$\mathbf{a}_i = -G \sum_{j=1,\neq i}^{N} m_j \frac{\mathbf{r}_i - \mathbf{r}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3},$$

and so the jerk is

$$\dot{\mathbf{a}}_i = -G \sum_{j=1,\neq i}^{N} m_j \left(\frac{\mathbf{v}_i - \mathbf{v}_j}{|\mathbf{r}_i - \mathbf{r}_j|^3} - 3 \frac{(\mathbf{v}_i - \mathbf{v}_j).(\mathbf{r}_i - \mathbf{r}_j)}{|\mathbf{r}_i - \mathbf{r}_j|^5} (\mathbf{r}_i - \mathbf{r}_j) \right).$$

Basic structure of an *N*-body code

1. Initialisation of $\mathbf{r}_i, \mathbf{v}_i, tnext_i$ (update time = $t_i + \Delta t_i$), $\mathbf{a}_i, \dot{\mathbf{a}}_i$, all i.
2. Choose i minimising $tnext_i$
3. Extrapolate all $\mathbf{r}_j, \mathbf{v}_j$ to $tnext_i$
4. Compute new $\mathbf{a}_i, \dot{\mathbf{a}}_i$
5. Correct new $\mathbf{r}_i, \mathbf{v}_i$ (Hermite integrator)
6. Compute new $tnext_i$
7. Repeat from step 2

Notes

- this does not include block time steps
- this is the basic structure of NBODY1, except for the choice of integrator. (The integrator is described in Bodenheimer et al).

Some Simple *N*-Body Codes with Variable Individual Time Steps

1. Binney & Tremaine 1e, 1987, PUP, Appendix 4.B (fortran)
2. Heggie & Hut, *The Gravitational Million-Body Problem*, 2004, CUP Appendix A, (matlab, C)
3. Hut & Makino, *The Art of Computational Science* (ruby) http://www.artcompsci.org/
4. http://www.nbabel.org/codes (C++,CUDA,F95,IDL,Java,PerlDL,PyC++,Python,R,etc.)

Note: none of these codes would reach beyond core collapse in a reasonable time, because of the appearance of binary stars at that point.

Quality control

How do you know that the results of a simulation are correct?

- There are no useful exact solutions
- Conserved quantities:
  1. Momentum
  2. Angular Momentum
  3. Energy.

  The most sensitive is energy.

  Output from NBODY1:

  T = 1.4 Q = 0.23 STEPS = 486 **DE = 0.000000 E = -0.250000**

  TC = 0.5

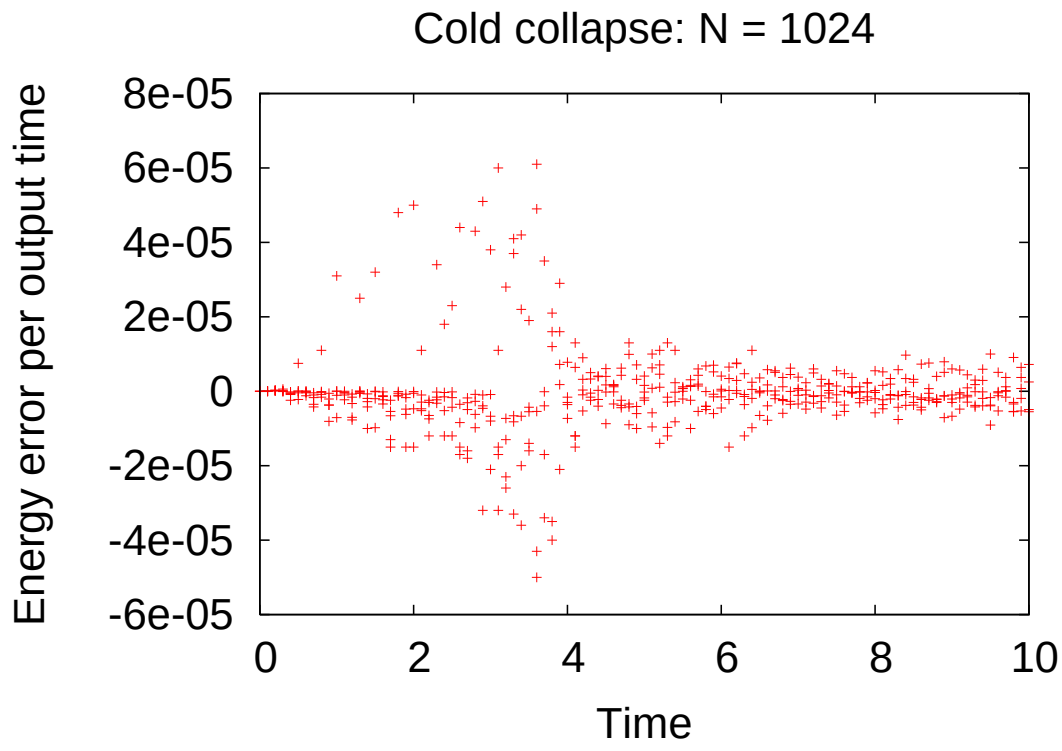
  <R> = 1.53 RCM = 0.0000 VCM = 0.0000 AZ = 0.00000 T6 = 4 NRUN = 1

  BINARY 6 15 0.040 0.040 -0.2 0.1840 3.6 0.0741 1.50 0.861 0

  BINARY 8 9 0.040 0.040 -0.2 0.2517 2.2 0.1231 1.84 0.979 0

Quality control (continued)

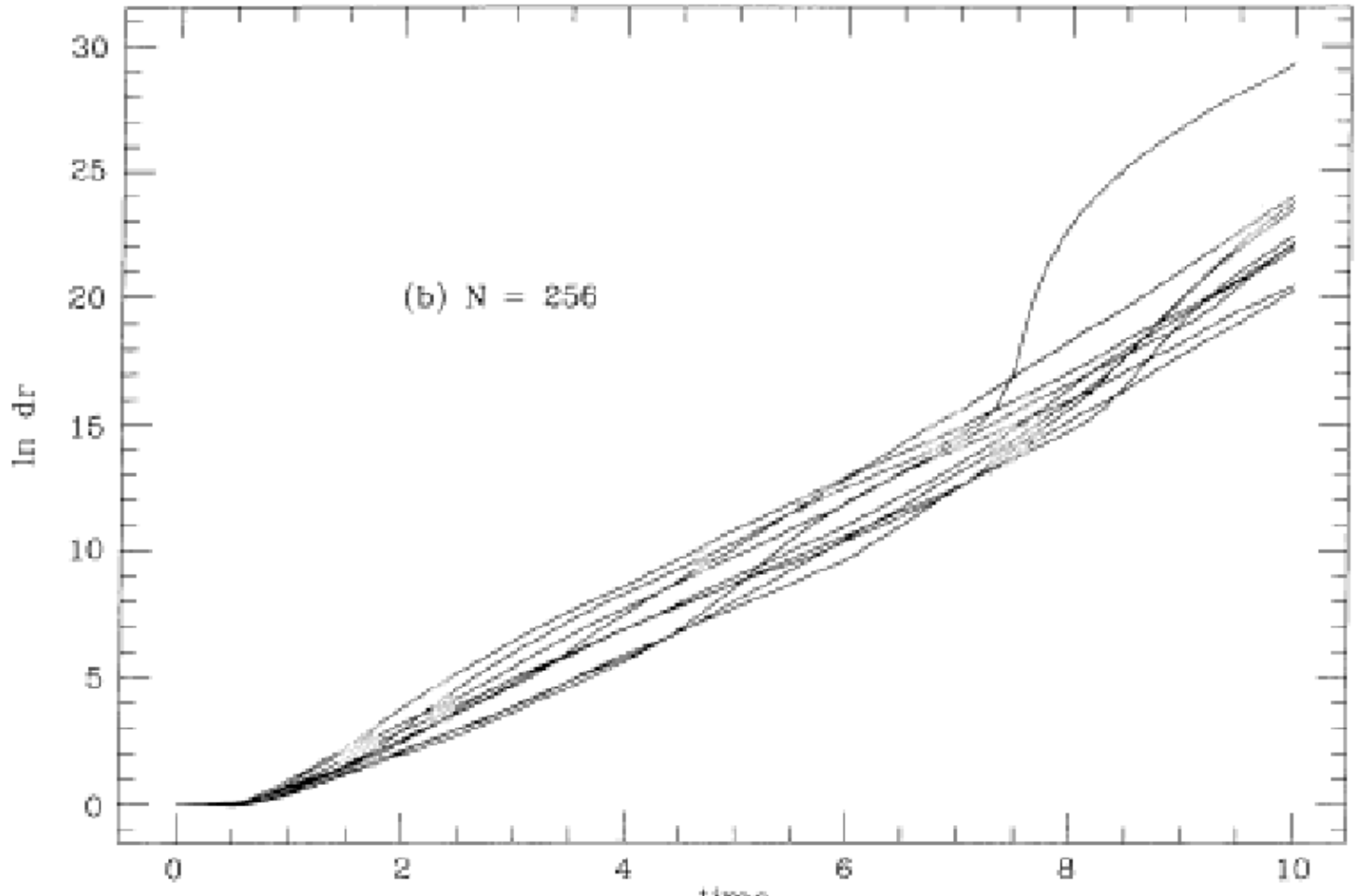
- (continued) Change in energy per output time



- Are your answers reasonable?

Growth of Errors in *N*-body simulations

Experiment: start two *N*-body simulations with slightly different initial conditions (say, one coordinate of one particle differing by 10^{-14}). How fast do the differences grow?



Growth of Errors in N-body simulations (continued)

- The growth of errors is exponential
- The time scale (in units of the crossing time) varies very slowly with N
- For a Plummer model, with equal masses, it is of order $0.1 t_{cr}$.
- Growth of error approximately 10^{18} when $\exp(t/(0.1 t_{cr}) = 10^{18}$, i.e. $t \simeq 4 t_{cr} \simeq 12 N$-body units
- After a short time the positions and velocities of particles are wrong
- We *assume* that the statistical properties of the simulation are correct

"Complexity"

Recall structure:

1. Initialisation of $\mathbf{r}_i, \mathbf{v}_i, tnext_i$ (update time = $t_i + \Delta t_i$), $\mathbf{a}_i, \dot{\mathbf{a}}_i$, all i. **Done once**
2. Choose i minimising $tnext_i$ **Proportional to** N
3. Extrapolate all $\mathbf{r}_j$ to $tnext_i$ **Proportional to** N
4. Compute new $\mathbf{a}_i, \dot{\mathbf{a}}_i$ **Proportional to** N
5. Correct new $\mathbf{r}_i$, compute new $\mathbf{v}_i$ (Hermite integrator)
6. Compute new $tnext_i$
7. Repeat from step 2

For each time step Δt, the computational effort is approximately proportional to N.

"Complexity" (continued)

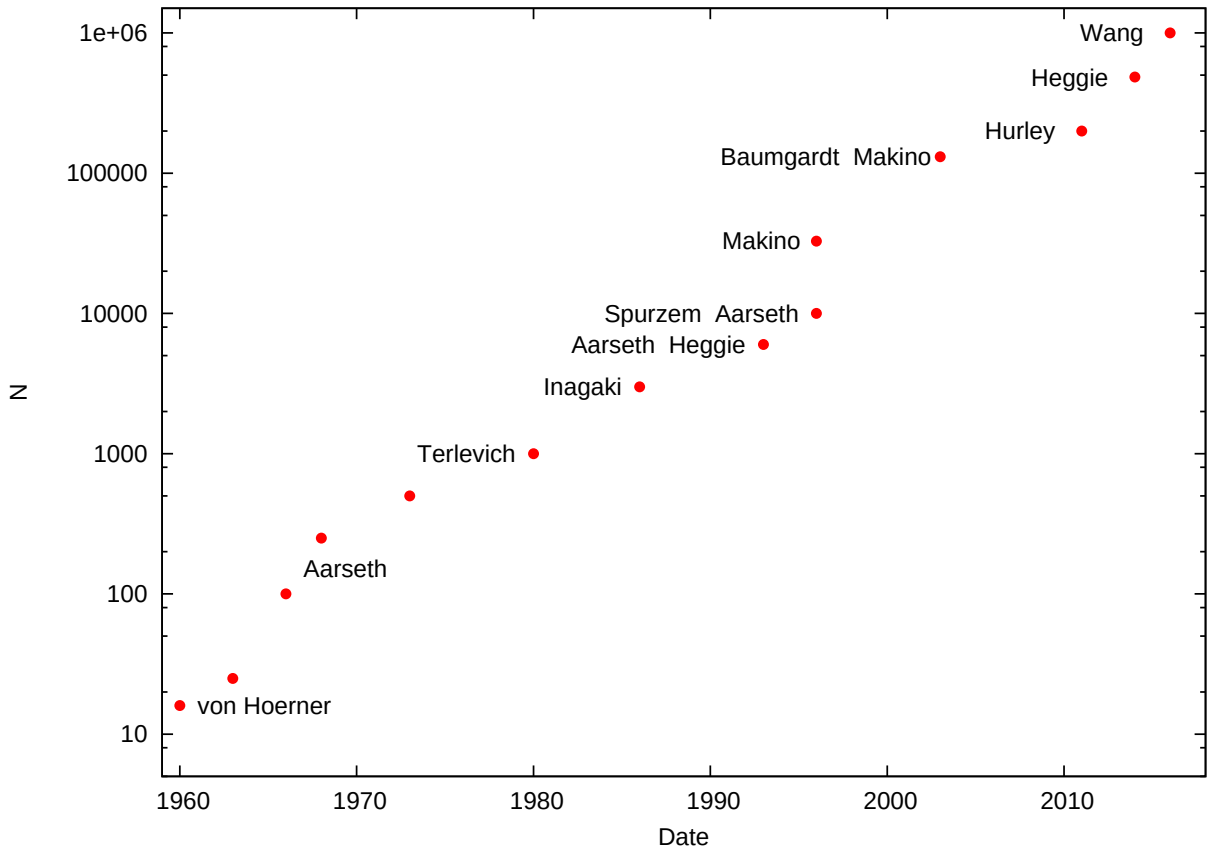
Typical time step is $\Delta t \sim \frac{r}{v}$, where v is typical speed and r is typical distance to nearest neighbour, i.e. $R/N^{1/3}$, where R is the virial radius. In Hénon (N-body) units (Lecture 1) $v \sim 1$, $R = 1$, and so $\Delta t \sim N^{-1/3}$.

Each time step takes of order N operations, and there are N particles. Hence the computational effort per N-body time unit is of order $N.N.N^{1/3} = N^{7/3}$.

Core collapse takes a few relaxation times, and $\frac{t_{rh}}{t_{cr}} \propto N/\ln \gamma N$ (Binney & Tremaine,Sec.1.2.1), where $t_{cr} = 2\sqrt{2}$. Hence the computational effort to core collapse varies as $N^{10/3}/\ln \gamma N$, or roughly N^3.

The Development of *N*-body Simulations in History

Though computers have doubled in speed every 18 months until recently (Moore's Law), they have doubled in *N* much more slowly: about a factor 10 every 10 years.



The Challenge of Milky Way Globular Clusters for *N*-body

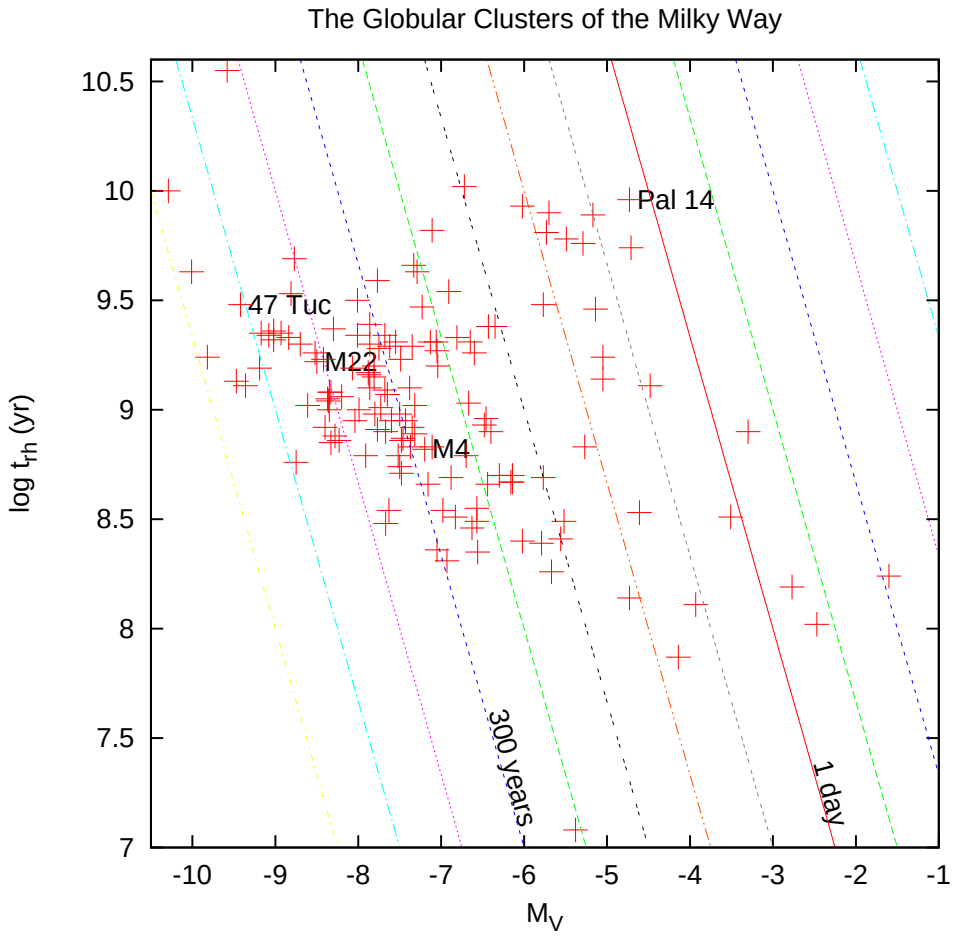


Figure : Contour lines of computational effort at intervals of 1dex

Three Bottlenecks in Each Time Step

The following stages are proportional to N in the basic structure of an N-body code:

- Choose i minimising $tnext_i$ **Can be made proportional to $N^{1/2}$ (see NBODY1)**
- Extrapolate all $\mathbf{r}_j$ to $tnext_i$ **Can be reduced by use of "block time steps"** (mentioned earlier in this lecture)
  - Group together particles which have very similar update times. The extrapolation is shared among them. Still proportional to N, but much smaller importance.
- Compute new $\mathbf{a}_i, \dot{\mathbf{a}}_i$ **This is the main residual bottleneck**
- Software solutions:
  - Neighbour scheme; incorporated in NBODY2
  - Tree codes (hierarchical schemes): see Bodenheimer et al
  - Grid-based methods, e.g. finite differences

Accelerating force calculation: hardware

1. Parallel computation. Example (code fragment of NBODY1, showing calculation of force on particle I)

```
DO 10 J = 1,N
   IF (J.EQ.I) GO TO 10
   A1 = X(1,J) - XI
   A2 = X(2,J) - YI
   A3 = X(3,J) - ZI
   RIJ2 = A1*A1 + A2*A2 + A3*A3 + EPS2
   A5 = BODY(J)/(RIJ2*SQRT(RIJ2))
   FIRR(1) = FIRR(1) + A1*A5
   FIRR(2) = FIRR(2) + A2*A5
   FIRR(3) = FIRR(3) + A3*A5
10 CONTINUE
```

The calculation of A1*A5, A2*A5, A3*A5 can be done simultaneously for different particles J, because there is no data dependency: the result of one calculation does not affect any other.

Accelerating force calculation: hardware (continued)

Parallel computers (continued)

- Calculations for different J-particles can be farmed out to different processors on the one parallel computer, and/or different cores on a single processor.
- For supercomputers or large clusters, best code is NBODY6++ (https://github.com/lwang-astro/betanb6pp/branches)
- For low-budget users, best option is video cards: Graphics Processing Unit (GPU)

- £200 (5000 CZK) each, speedup x100
- Code: NBODY6 + GPU2 (http://www.ast.cam.ac.uk/~sverre/web/pages/nbody.htm)

Software refinements

Close encounters and few-body subsystems

Two problems when particles come very close together:

1. Subtraction of positions of two close neighbours causes an increase in relative rounding error
2. Reduced time step, and the simulation may slow down dramatically.

Close encounters and binaries. I. Offset "regularisation"

Suppose particles i, j form a bound pair, or experience a close encounter. Use offset variables (Jacobi coordinates) $\mathbf{r}, \mathbf{R}$ defined as

$$\begin{aligned} \mathbf{R} &= \frac{m_i \mathbf{r}_i + m_j \mathbf{r}_j}{m_i + m_j} \\ \mathbf{r} &= \mathbf{r}_i - \mathbf{r}_j, \end{aligned}$$

and write equations of motion in terms of $\mathbf{r}, \mathbf{R}$: e.g.

$$\ddot{\mathbf{r}} = -G(m_i + m_j)\frac{\mathbf{r}}{|\mathbf{r}|^3} + \mathbf{a}'_i - \mathbf{a}'_j,$$

where $'$ means we omit force due to i, j.
Advantage: avoids rounding error in repeated calculation of $\mathbf{r}_i - \mathbf{r}_j$.
Note: $\mathbf{R}$ is centre of mass position vector, $\mathbf{r}$ is position of particle j relative to particle i

Close encounters and binaries. II. KS regularisation

- Singularity in

$$\ddot{\mathbf{r}} = -G(m_i + m_j)\frac{\mathbf{r}}{|\mathbf{r}|^3} + \mathbf{a}'_i - \mathbf{a}'_j$$

  requires small time steps for close and/or eccentric binaries.
- KS regularisation is subtle change of variables which removes the singularity.

Example: one-dimensional regularisation

Unperturbed binary motion is

$$\ddot{x} = -\frac{1}{x^2}$$

where we have scaled to units such that $G(m_i + m_j) = 1$.
Introduce new variables z, τ (transformed coordinate and time) such that

$$\begin{aligned} x &= z^2 \\ \frac{dt}{d\tau} &= x \end{aligned}$$

One-dimensional regularisation (continued)

Since $x = z^2$, $\dot{x} = 2z\dot{z}$.
Since $dt/d\tau = x$,

$$\begin{aligned} \dot{x} &= 2zz' \frac{d\tau}{dt} \text{ where } ' \text{ means } d/d\tau \\ &= \frac{2zz'}{x} \text{ since } dt/d\tau = x \\ &= \frac{2z'}{z}. \end{aligned}$$

Differentiating again with repect to time gives similarly

$$\begin{aligned} \ddot{x} &= \frac{2z''}{z^3} - 2\frac{z'^2}{z^4} \\ &= -\frac{1}{x^2} \text{ (equation of motion)} \\ &= -\frac{1}{z^4} \end{aligned}$$

One-dimensional regularisation (continued)

We have

$$\frac{2z''}{z^3} - 2\frac{z'^2}{z^4} = -\frac{1}{z^4}$$

Hence

$$\begin{aligned} z'' &= \frac{z'^2}{z} - \frac{1}{2z} \\ &= \frac{1}{2}z\left(\frac{2z'^2}{z^2} - \frac{1}{z^2}\right) \\ &= \frac{1}{2}z\left(\frac{1}{2}\dot{x}^2 - \frac{1}{x}\right) \\ &= \frac{1}{2}hz, \end{aligned}$$

where

$$h = \frac{1}{2}\dot{x}^2 - \frac{1}{x}.$$

One-dimensional regularisation (continued)

Now

$$h = \frac{1}{2}\dot{x}^2 - \frac{1}{x}$$

is just the energy of the binary (per unit [reduced] mass); the kinetic energy is $\dot{x}^2/2$, the potential energy is $-1/x$. Thus in the transformed equation of motion

$$z'' = \frac{1}{2}hz,$$

the coefficient of z is constant. This is the simple harmonic oscillator equation.

Bottom line: we have transformed the one-dimensional Kepler problem

$$\ddot{x} = -\frac{1}{x^2}$$

(which is singular at $x = 0$) into the simple harmonic oscillator equation, which is regular everywhere.

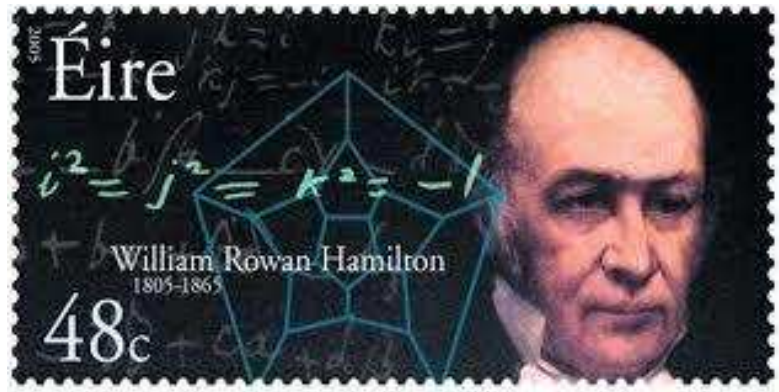
Two-body regularisation: practicalities

Recall: regularisation is the process which transforms the Kepler equation of motion into the simple harmonic oscillator equation. It was established in the one-dimensional case.

- Need a three-dimensional version. This exists and is called KS (or quaternion) regularisation.
- Candidates for regularisation recognised by short time steps
- Deregularisation if the binary is too strongly perturbed
- Still requires short time step for close binary
- Freeze unperturbed binaries
- Code: NBODY3

KS regularisation

- quaternions are a generalisation of complex numbers
- A quaternion is written as $a + ib + cj + dk$, where the units i, j, k satisfy $i^2 = j^2 = k^2 = -1$ and $ijk = -1$. Hence $-i = i^2jk = -jk$ and, more generally, $ki = j, ij = k$. Their arithmetic is non-commutative.
- Every quaternion has a *conjugate* defined to be $a - ib - jc - kd$.
- Three-vectors $\mathbf{r} = (x, y, z)$ can be mapped to the pure imaginary quaternions $ix + jy + kz$



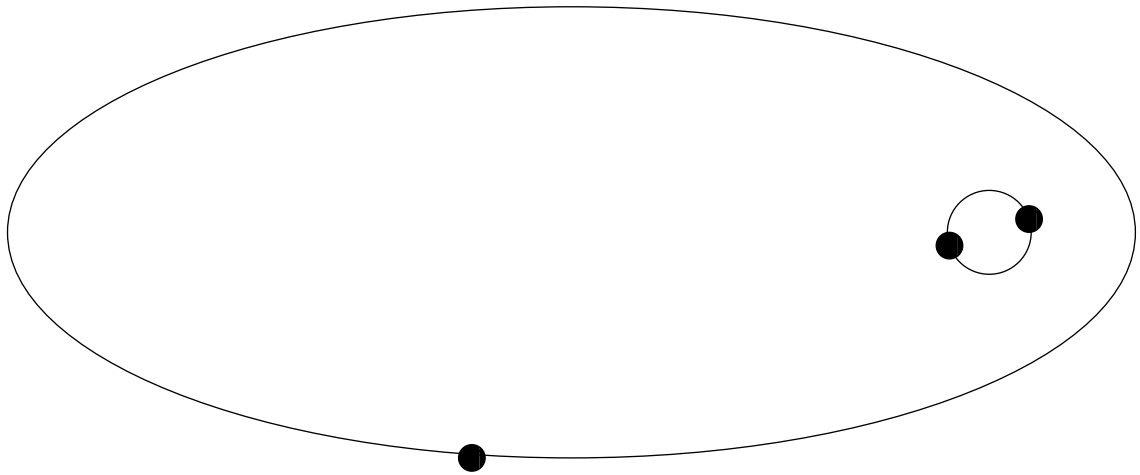
- Any three-vector (written as a quaternion) can be expressed as $ix + jy + kz = qi\bar{q}$ for some quaternion q.
- This is the generalisation of the one-dimensional transformation $x = z^2$.

KS regularisation (contd)

- The same time-transformation $\frac{dt}{d\tau} = r$ is used
- A further assumption that $\dot{q} i \bar{q} = q i \dot{\bar{q}}$ is made
- Then the equation of Kepler motion $\frac{d^2\mathbf{r}}{dt^2} = -\frac{Gm\mathbf{r}}{r^3}$ transforms again to $\frac{d^2 q}{d\tau^2} = \frac{h}{2} q$.
- See Heggie & Hut, ch.15
- In practice, the transformations are written in terms of spinors (2×2 complex matrices) or 4×4 real matrices

Higher-order subsystems: triples, quadruples, etc

- hierarchical triples are binaries constantly perturbed by a third body: there is a procedure called “slow-down” which follows secular perturbations with (much) larger time step



- non-hierarchical triples, quadruples: *chain regularisation*, a generalisation of offset and KS regularisation; there are specilisations to triples and quadruples; codes TRIPLE, CHAIN on Aarseth’s download page

Exercise for Lecture 3

1. Examine the output of a run of NBODY1 which completed successfully. The last line gives “CPUTOT”, which is the execution time in minutes. (The data may also be obtained using the unix command “time”.) The last main output will also give the number of integration steps completed (“STEPS”). Repeat for larger values of N. (If you find that a run terminates quickly with “FATAL ERROR! BAD INPUT N = 250” (or something like that) go to the Real8 subdirectory, edit the parameter NMAX in params.h, and recompile.) Can you understand the dependence of the computing time on N, assuming that the computational effort is dominated by the force calculation?