DRAFT VERSION MAY 11, 2022
Typeset using LaTeX **twocolumn** style in AASTeX631

# The Kendrick N-body Simulator

ROBEL GEDA [1]

[1] *Department of Astrophysical Sciences, Princeton University, Princeton, NJ 08540, USA*

(Received May 10, 2022)

ABSTRACT

N-body simulations play an important role in probing and modeling the dynamical evolution of various celestial systems. In this project, I explore N-body simulations by implementing a CUDA-based N-body gravity solver and demonstrate its capabilities. Kendrick (**K**inetics **E**xploration and **N**-body **D**ynamics **R**esearch **I**nvolving **C**UDA **K**ernels) is a code to perform and explore N-body simulations. It can be used to construct initial conditions, compute gravitational forces, and visualize results using a third-party software called ParaView (Ahrens et al. 2005). The full package can be found and downloaded from the `robelgeda/kendrick_nbody` GitHub repository.

## 1. INTRODUCTION

N-body simulations iteratively model the dynamical evolution of particles under influence of forces. For this project, I will be focusing on point particles (which can be smoothed) under their mutual gravitational attraction. At each time step of the simulation, the dynamical interaction of each particle (particle $i$) is computed using Newton's law of gravity:

$$\frac{d^2\vec{x}_i}{dt^2} = -G\sum_{j=0\neq i}^{N-1} m_j \frac{\vec{x}_i - \vec{x}_j}{|\vec{x}_i - \vec{x}_j|^3} \tag{1}$$

Where $G$ is Newton's gravitational constant, $N$ is the number of particles, $i$ and $j$ are the particle indices (Python based), and $x$ is the Cartesian coordinates of the particles.

CUDA (**C**ompute **U**nified **D**evice **A**rchitecture) is a parallel computing platform and programming language that enables the use of graphics processing units (GPUs) for general-purpose computation. GPUs contain hundreds of processing cores that can be used to effectively distribute computational tasks, reducing run times. Though N-body simulations can not be parallelized across time steps, GPUs can help speed up the simulations by parallelizing the gravitational calculations for each particle across many compute cores.

Kendrick started as a personal undergraduate project to gain familiarity with the CUDA programing language in preparation for an unrelated research project at the Rutgers Blueprint Research Lab (Computer Science). The initial version of this code was able to distribute a user-defined function to multiple GPU cores using CUDA. In this project, I implement functions to accurately compute gravitational interactions, evolve particle positions, and estimate their energies. Much of this paper is a concise description of these functions and their results. This software is written in C++ but I will use Python syntax for the pseudo code in the sections that follow for clarity.

## 2. SIMULATION CODE

In this section, I outline functions that compute gravitational forces, evolve particles, and compute energies. For clarity, I start with a regular N-body code in section 2.1 and describe the paralyzed version in section 2.2.

### 2.1. *N-body Code*

As described in the introduction, the primary objective of an N-body code is to evolve the positions of particles at each timestep using the computed accelerations. As such, the code must iterate through each particle in the simulation and compute the gravitational acceleration caused by every other particle in the simulation. This can be best implemented using a `for` loop as follows:

**Algorithm 1.** Acceleration and Potential

```
# time-step t
for i in particles:
    for j in particles:
        if i == j: skip
        a[i] += calc_acceleration(i, j)
        pot[i] += calc_potential(i, j)
```

Where `calc_acceleration` computes the acceleration using a modified version of Equation 1, which adds a softening factor ($b$) to avoid divergence due to close encounters:

$$\frac{d^2\vec{x}_{ij}}{dt^2} \approx -Gm_j \frac{\vec{x}_i - \vec{x}_j}{(|\vec{x}_i - \vec{x}_j|^2 + b^2)^{3/2}} \tag{2}$$

And the `calc_potential` function computes the gravitational potential (i.e potential energy per unit mass):

$$\Phi_{ij} \approx \frac{-Gm_j}{(|\vec{x}_i - \vec{x}_j|^2 + b^2)^{1/2}} \tag{3}$$

After the accelerations of all of the particles are known, their positions can be evolved using yet another `for` loop:

**Algorithm 2.** Evolving Positions

```
# time-step t
for i in particles:
    v[i] = calc_velocity(a[i])

    # Update for time-step t+1
    x[i] = calc_position(x[i], v[i])
```

Where `calc_velocity` computes the velocity of the current timestep as follows:

$$\vec{v}_{i,t} \approx \vec{v}_{i,t-1} + \frac{d^2\vec{x}_i}{dt^2}\Delta t \tag{4}$$

Where $\Delta t$ is the timestep size (interval between timesteps). Once the velocity is computed, it can be used to evolve the positions of the particles to the next time step:

$$\vec{x}_{i,t+1} \approx \vec{x}_{i,t} + \vec{v}_{i,t}\Delta t \tag{5}$$

### 2.2. *Parallelized N-body Code*

At each timestep, it is possible to parallelize the acceleration computation since each pairwise interaction can be computed independently. The algorithm in Section 2.1 has a hefty computational complexity of $\mathcal{O}(n^2)$. But if we assign each GPU thread with a particle and compute the acceleration for that particle, we substantially improve the algorithm's complexity to an order of $\mathcal{O}(n)$. This is because the interaction of each particle is taken care of simultaneously by the hundreds of thousands of GPU threads. As such, I replaced the first for loop in Algorithm 1 with a function that can be executed by each GPU thread. I use the GPU thread ids as indices that represent the particle that the thread is assigned.

**Algorithm 3.** GPU Acceleration and Potential

```
# time-step t
def gpu_thread():
    i = thread.id
    for j in particles:
        if i == j: skip
        a[i] += calc_acceleration(i, j)
        pot[i] += calc_potential(i, j)
```

## 3. PLUMMER TEST

In this section, I demonstrate Kendrick by simulating a globular cluster that I model using a Plummer sphere. In the following sub-sections, I describe the initialization processes and energy tests to assess the simulation.

### 3.1. *Model Initialization*

To initialize an N-body simulation means to set the initial masses, positions, and velocities of each particle in the simulation. To do this, we start with Plummer's model (or Plummer model). The Plummer model is defined by a softened gravitational potential. The Plummer potential can be described as follows (in polar coordinates):

$$\Phi = \frac{-GM}{(r^2 + a^2)^{1/2}} \tag{6}$$

Where $M$ is the total mass of the system and $r$ is the distance from the center of the model. It is possible to use $r = |\vec{r}|$ in this case because the model is isotropic. Given $\Phi$ and by defining all particles to have the same masses, we can define the initial positions and velocities.

#### 3.1.1. *Particle Mass*

Because I define the mass of all particles to be the same, the mass of each particle is simply the total mass divided by the number of particles:

$$m = \frac{M}{N} \tag{7}$$

#### 3.1.2. *Positions*

In Section 3.1 we discussed the isotropic nature of the Plummer model used for this test. I defined the Plummer model in polar coordinates thus it is also necessary to translate the coordinates into Cartesian coordinates. To define the position of a particle, we need three parameters $(r_i, \theta_i, \phi_i)$. $\theta_i$ and $\phi_i$ can be assigned by drawing from a uniform distribution since the model is isotropic. In order to properly draw the radius however, we must take the density of the system into consideration. The density of the model can be found by solving the Poisson equation:

$$\nabla^2\Phi = -4\pi G\rho \tag{8}$$

$$\frac{1}{r^2}\frac{d}{dr}\left(r^2\frac{d\Phi}{dr}\right) = \frac{1}{r^2}\left(2r\frac{d\Phi}{dr} + r^2\frac{d^2\Phi}{dr^2}\right) = -4\pi G\rho \tag{9}$$

To solve this, I take the first and second derivatives of $\Phi$:

$$\frac{d\Phi}{dr} = GM\frac{r}{(r^2 + a^2)^{3/2}} \tag{10}$$

$$\frac{d^2\Phi}{dr^2} = GM\frac{a^2 - 2r^2}{(r^2 + a^2)^{5/2}} \tag{11}$$

Solving for $\rho$ in Equation 9 and using the first and second derivatives, the density can be defined as:

$$\rho(r) = \frac{M}{4\pi}\frac{3a^2}{(r^2+a^2)^{5/2}} \tag{12}$$

And the mass enclosed within a given radius can be defined as:

$$M(<r) = \int_0^r 4\pi r^2 \rho(r) dr = M\frac{r^3}{(r^2+a^2)^{3/2}} \tag{13}$$

Using Equation 13, the radius can be randomly drawn by drawing an enclosed mass from a uniform distribution and finding the radius that is associated with that enclosed mass. For simplicity I define the probability of having an enclosed mass ($P$) as follows:

$$P = \frac{M(<r)}{M} = \frac{r^3}{(r^2+a^2)^{3/2}} \tag{14}$$

I invert this probability by solving for $r$. The radius drawn from a uniform distribution $P$ is thus:

$$r(P) = \frac{a}{(P^{-2/3}-1)^{1/2}} \tag{15}$$

With this, it is possible to define $(r_i, \theta_i, \phi_i)$ for each particle in the simulation. I convert the randomly drawn positions to Cartesian coordinates using the standard transform.

#### 3.1.3. *Velocities*

To initialize the velocities of the particles, I again start from spherical coordinates $(v_i, \theta_i, \phi_i)$. Given the isotropic nature of the system, we can again draw $\theta_i$ and $\phi_i$ from a uniform distribution. The velocity magnitude on the other hand needs to be derived from the systems phase-space distribution function. I was able to derive the radius without the consideration of the distribution function because the density of the system is a projection of the phase-space distribution function $f(v)$:

$$\rho(r) = 4\pi \int_0^{v_{esc}} f(v) v^2 dv \tag{16}$$

It is more convenient to describe this in terms of energy per unit mass ($E$). To do so, I define $v_{esc}$ as:

$$E = \frac{1}{2}v_{esc}^2 + \phi = 0 \Rightarrow v_{esc} = (-2\Phi)^{1/2} \tag{17}$$

And $v^2 dv$ as follows:

$$E = \frac{1}{2}v^2 + \phi \Rightarrow v = (2E - 2\Phi)^{1/2} \tag{18}$$

$$dv = (2E-2\Phi)^{-1/2} dE \tag{19}$$

$$v^2 dv = (2E-2\Phi)^{1/2} dE \tag{20}$$

Adding these expressions into Equation 16, the following expression is obtained:

$$\rho(r) = 4\pi \int_{(-2\Phi)^{1/2}}^0 f(E)(2E-2\phi)^{1/2} dE \tag{21}$$

Since $E$ and $\Phi$ are negative, Binney & Tremaine (2008) defines a new set of positive variables to make the integration easier. They define $\mathcal{E} = -E$ and $\Psi = -\Phi$, which yields the following Ergodic distribution function:

$$\rho(r) = 4\pi \int_0^{\Psi} f(\mathcal{E})(2\mathcal{E} - 2\psi)^{1/2} d\mathcal{E} \tag{22}$$

Note that for this integration using $\Psi$ as upper bound is permitted because $f(E < (-2\Phi)^{1/2}) = 0$. Binney & Tremaine (2008) substitute $\rho(\Psi) = \frac{3}{4\pi}\Psi^5$ and use an Abel integral inversion to solve for $f(\mathcal{E})$ which yields Eddington's formula:

$$f(\mathcal{E}) = \frac{1}{\sqrt{8}\pi^2}\left[\int_0^{\mathcal{E}} \frac{d^2\rho}{d\Psi^2}\frac{d\Psi}{\sqrt{\mathcal{E}-\Psi}} + \frac{1}{\mathcal{E}}\left(\frac{d\rho}{d\Psi}\right)_{\Psi=0}\right] \tag{23}$$

For the Plummer model, which is a polytrope with $n = 5$, the distribution function becomes:

$$f(\mathcal{E}) = \frac{24\sqrt{2}}{7\pi^3}\frac{a^2}{G^5 M^4}\mathcal{E}^{7/2} \tag{24}$$

Following Aarseth et al. (1974) Appendix A, I define the energy in terms of $q = \frac{v}{v_{esc}}$ at fixed radii as:

$$\mathcal{E}(r,v) == \frac{1}{2}v_{esc}^2(r) - \frac{1}{2}v^2 = \frac{1}{2}v_{esc}^2(r)(1-q^2) \tag{25}$$

At a constant radius, $v_{esc}^2(r)$ is a constant, which means $\mathcal{E}(q) \propto 1-q^2$. Thus, at a given radius, the probability function $f(q)$ is proportional to the function $g(q)$ as follows:

$$g(q) = q^2(1-q^2)^{7/2} \tag{26}$$

Equation 26 is not straight forward to invert. Thus, following Aarseth et al. (1974), I use a von Neumann technique to invert equation 26 and find the magnitude of the velocity. The range for $q$ values is 0 to 1 and the range for $g(x)$ is 0 to 0.1 (Aarseth et al. 1974). With this method, it is possible to obtain randomly drawn $(v_i, \theta_i, \phi_i)$, which I convert to Cartesian coordinates using the standard transform.

### 3.2. *Simulation and Energy Test*

To test the code, I run a simulation that represents a $10^5 M_\odot$ globular star cluster that spans a Plummer radius of $50ly$. I initialize a Plummer sphere with 100,000 particles at a mass resolution of $1M_\odot$ per particle (see Figure 1).

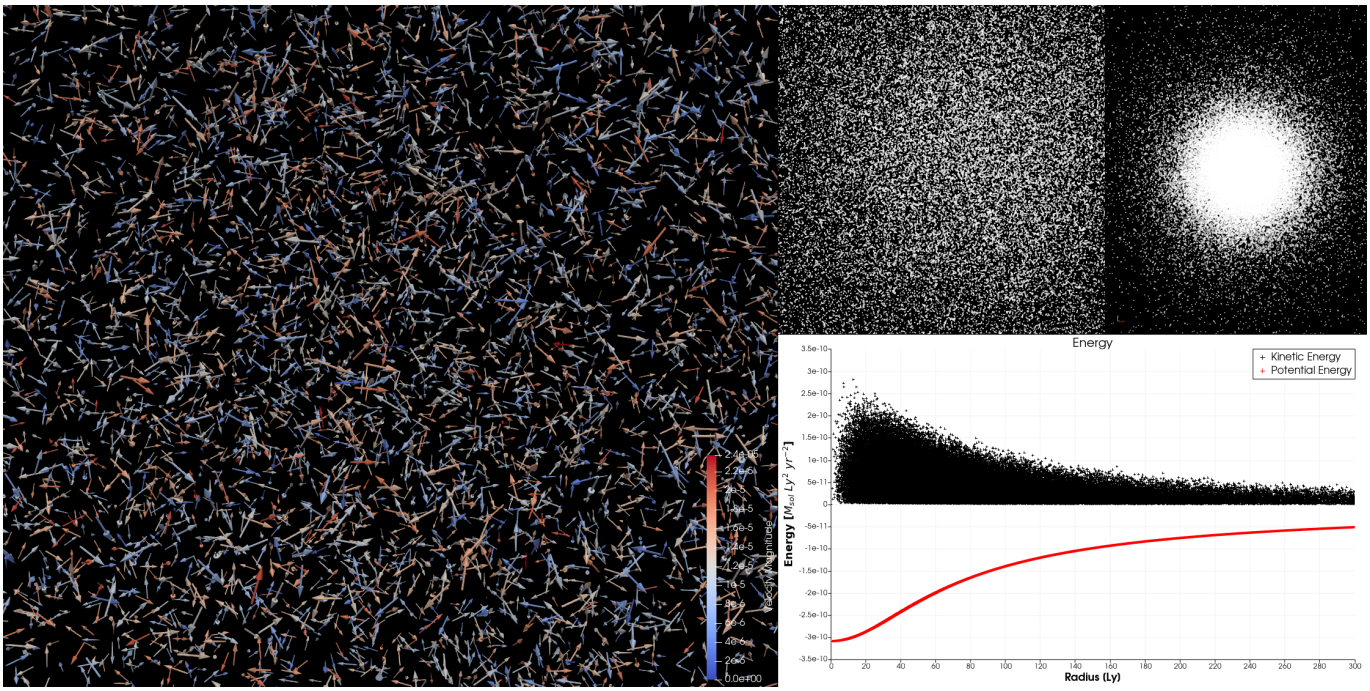


**Figure 1.** Initial conditions of the simulation. The first panel shows the velocity and spacial distributions of stars at the center of the cluster. Colors in the fist panel represent velocity magnitudes and the arrows the direction of the velocity. The two top right panels show the cluster's point distributions. The bottom right panel shows the kinetic and potential energies.

For the remainder of this section, I will be using units of light-years for length, years for time, and solar masses for mass. It is common practice to normalize the gravitational constant $G$ to equal one but I chose not to do so in favor of the physical intuition that can be gained from working with the units I have described here. Thus, the units of the simulation are set by defining $G$ as stated in Table 1.

**Table 1.** Simulation Parameters

| Parameter | Value |
|---|---|
| G | $1.5607939 \cdot 10^{-13}\ ly^3\ M_\odot^{-1}\ y^{-2}$ |
| N | $10^5$ |
| M | $10^5\ M_\odot$ |
| a | $50\ ly$ |
| b | $3\ ly$ |
| $\Delta t$ | 10 y |
| Num timesteps | $10^6$ |
| Simulation time | $10^7\ y$ |

The simulation was carried out on Princeton's Adroit cluster which boasts a powerful NVIDIA A100 GPU. The simulation finished just under 24 hours with over 90% CPU efficiency.

### 3.2.1. *Energy Test of Initial Conditions*

In this Section I test the energies of the initial conditions. The theoretical total potential of a Plummer sphere is given by (Aarseth et al. 1974):

$$U = \frac{-3\pi}{32}\frac{GM^2}{a} \tag{27}$$

And the kinetic ($K$) and total energy ($\hat{E}$) should be (Aarseth et al. 1974):

$$K = -\hat{E}_{tot} = -\frac{U}{2} \tag{28}$$

Table 2 shows the expected and measured energies of the initial conditions. The potential and kinetic energies were computed by summing the potential and kinetic energy per unit mass, respectively (I also account for double counting pairwise potentials):

$$U = \frac{1}{2}\sum_i^N m_i \phi_i \tag{29}$$

$$K = \frac{1}{2}\sum_i^N m_i |\vec{v}_i|^2 \tag{30}$$

**Table 2.** Initial Condition Energies

| Parameter | Theoretical | Simulation | Error (100%) |
|---|---|---|---|
| $U$ | $-9.194e-06$ | $-9.150e-06$ | 0.50% |
| $K$ | $+4.597e-06$ | $+4.620e-06$ | 0.51% |
| $\hat{E}_{tot}$ | $-4.597e-06$ | $-4.529e-06$ | 1.46% |

#### 3.2.2. *Energy Test of Simulation*

To test the consistency of the simulation across timesteps, I compare potential energies of the system to the initial conditions. I do this by over-plotting the energies of each particle across all timesteps (see Figure 2). I also compare the total energy of the system at every timestep to the first timestep by calculating the % error (see Figure 3). For reference the dynamical and crossing times of the system can be calculated as follows:

$$t_{dyn} \approx \frac{R}{V} \approx \frac{a}{V_{rms}} \approx 5.320e+06\ y \tag{31}$$

$$t_{cross} \approx \frac{N}{log(N)}\, t_{dyn} \approx 1.064e+11\ y \tag{32}$$

### 3.3. *Conclusion*

In this paper presents the Kendrick N-body simulator. In Section 3 we see that the code is able to carry out N-body simulations with small errors. Overall, the Kendrick project was successful in exploring various aspects of simulating dynamical systems. Future improvements could include Barnes–Hut algorithms and the addition of new potential models. Please see the following YouTube video for an animation of the Plummer test.

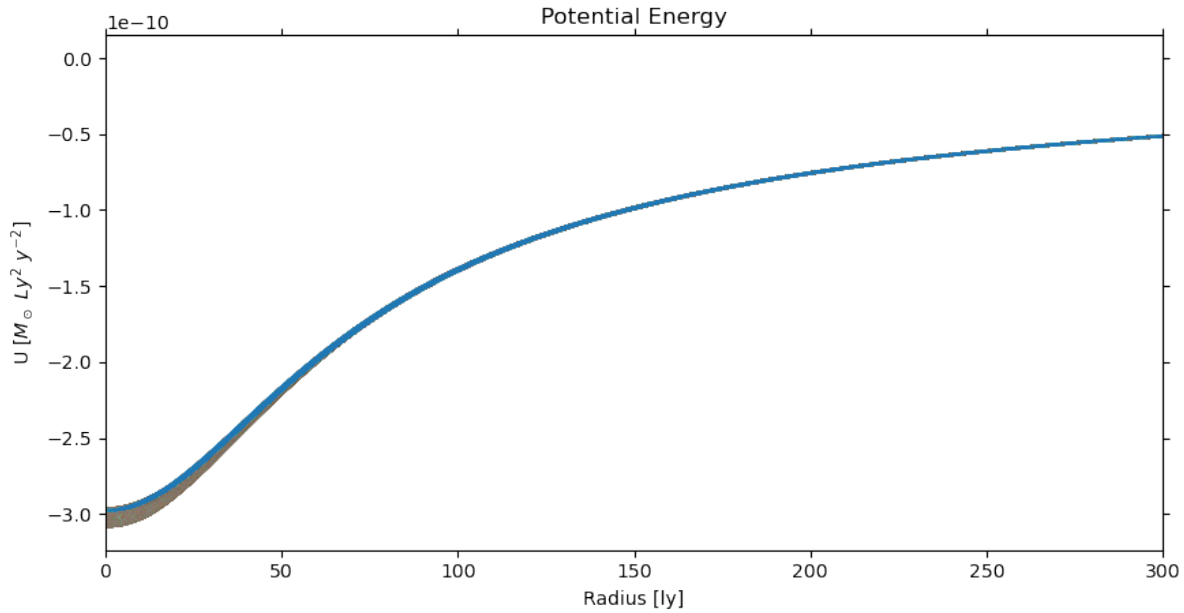


**Figure 2.** Potential energies of all 1000 timesteps over-plotted. There is divergence near the center of the cluster which is responsible for much of the errors coming from the potential energy. The plot spans a radius of $6a$.

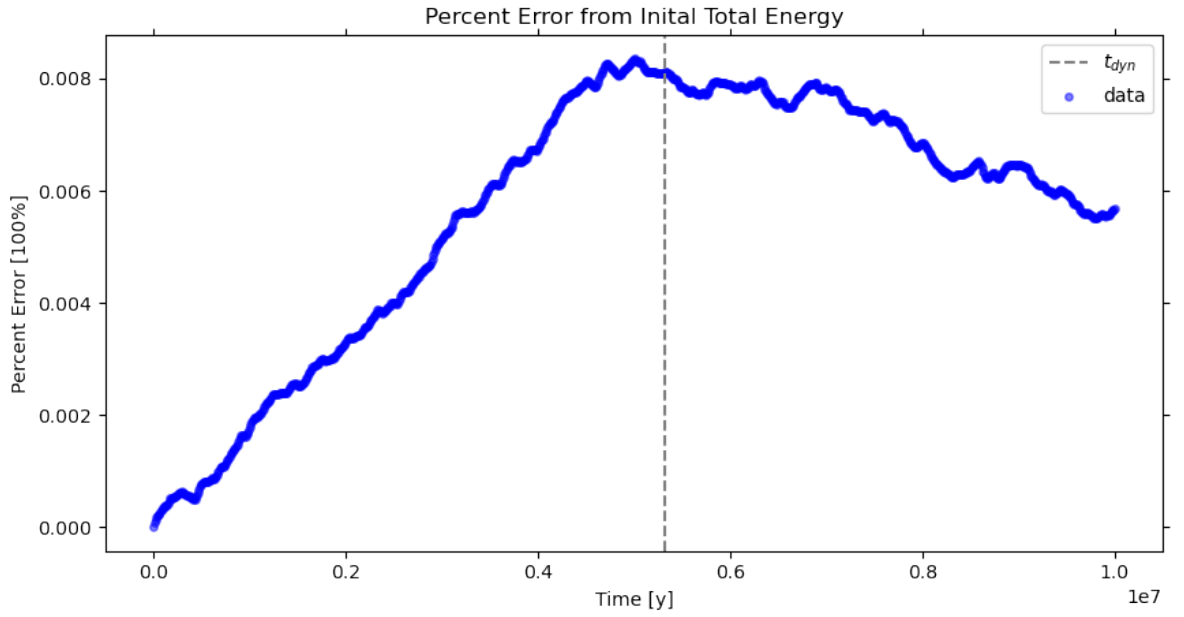


**Figure 3.** Scatter plot of percent errors of total energies compared to the initial condition as a function of time. The gray dashed line shows the dynamical time. This shows that errors cap off near the dynamical time. The errors are negligible across all time steps.

## REFERENCES

Aarseth, S. J., Henon, M., & Wielen, R. 1974, A&A, 37, 183

Ahrens, J., Geveci, B., & Law, C. 2005, in Visualization Handbook (Elsevier), 717–731

Binney, J., & Tremaine, S. 2008, Galactic dynamics, 2nd edn., Princeton Series in Astrophysics (Princeton, NJ: Princeton University Press)