UNIVERSITY OF CALIFORNIA,
IRVINE

The Oort Cloud Under the Influence of Dark Matter Halos

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Physics

by

Daniel Patrick Tierney

Thesis Committee:
Professor Asantha Cooray, Chair
Professor James Bullock
Professor Manoj Kaplinghat

2006

The thesis of Daniel Patrick Tierney is approved:

Committee Chair

University of California, Irvine
2006

DEDICATION

To

my parents,

Melissa Scholz,

and her parents

without whose loving and generous support

this project would not have been possible.

Thank you all

from the bottom of my heart.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank Prof. Asantha Cooray for generously taking me on as one of his graduate students, for giving me an interesting project, and for guiding me through the steps necessary to complete it.

I would also like to thank Evan Dowling for his suggestions on how to set up the model and on how to solve it numerically. His knowledge of programming far exceeds my own and he has generously made this knowledge available to me.

I am grateful that I was able to use many free CPU cycles on the desktop computers in the high energy physics group at UC Irvine for many months while I tested and ran my simulations. I am also grateful to Joseph Farran for helping me to set my code up to run on the MPC and GradEA Beowulf clusters on campus. Without the use of those machines I would not have been able to finish this project in a timely manner.

I would also like to thank Profs. Manoj Kaplinghat and James Bullock for sitting on my thesis committee, for reading my thesis, and for providing helpful feedback while I defended my thesis.

ABSTRACT OF THE THESIS

The Oort Cloud Under the Influence of Dark Matter Halos

By

Daniel Patrick Tierney

Master of Science in Physics

University of California, Irvine 2006

Professor Asantha Cooray, Chair

In a recent paper by J. Diemand, et. al., the authors proposed that dark matter could be clumped into halos with masses several orders of magnitude heavier than the sun to several orders of magnitude lighter. The authors further suggested that these halos may disrupt the Oort cloud of comets. We first simulate 1000 comets under the influence of the sun, Galactic disk, and Galactic bulge to model the Oort cloud as accurately as possible. Next we pass halos with masses between 0.001 $M_\odot$ and 10 $M_\odot$ by the Oort cloud, we measure the population of the Oort cloud as a function of time, and we measure the flux of comets into the solar system. Then we compare cases without halos to cases with halos to put constraints on the possible masses of Diemand's halos. In this paper we argue that heavy halos disrupt the Oort cloud enough that they can be ruled out, while light halos do not disrupt the Oort cloud enough for us to rule them out.

1 Introduction

In a recent paper by J. Diemand, et al., [3] (hereafter referred to as "Diemand") the authors presented "supercomputer simulations of the concordance cosmological model assuming neutralino dark matter" from which they found "the first objects to form are numerous earth mass dark matter halos about as large as the solar system." They further reported that these halos are stable and roughly 10^{15} could exist in the Milky Way at present. If such structures exist Diemand suggests they "may cause significant perturbations to some of the bodies orbiting in the Oort cloud surrounding the solar system." As the universe ages, Diemand's halos "merge successively into more massive systems," with possible masses ranging from the mass of Earth to several orders of magnitude heavier than the sun, and there are orders of magnitude more "light" halos than there are "heavy" halos.

We have found that heavy halos can gravitationally strip comets out of the Oort cloud when they pass by, with more massive halos stripping out more comets. Heavy enough halos can even dissipate the entire Oort cloud before the solar system reached its present age. This scenario conflicts with our observations since the Oort cloud is believed to exist. We therefore use the following procedure to place limits on the halos' masses.

First, we build a simulation that models the Oort cloud as accurately as possible. Next we pass halos with different masses by the Oort cloud and simulate the cloud's response to them. Then compare the population of the cloud as a function of time in cases where there are no halos to cases with halos. If halos with mass m remove too many comets then we can argue than Diemand's halos must be lighter than m. In this way we study in this paper whether the dynamics of the Oort cloud can be used as a novel means to put constraints on the possible masses of Diemand's halos.

We proceed by introducing some background on dark matter in §2 and some background on the Oort cloud in §3. We will use this information in §4 to construct a mathematical model of the Oort cloud with halos. We will discuss how we solved this model numerically in §5 and we will briefly discuss the details of the C++ program we used in §6. Finally, we will discuss our results in §7. Our computer code appears in Appendix A.

2 The nature of dark matter

A variety of observations have added to the theoretical motivations for the existence of dark matter (DM). In the paragraphs that follow we discuss one observation and list a few of the DM candidates, but it is not the purpose of this paper to convince the reader of the existence of DM or to argue what form it ought to take. Instead, we simply presume that DM exists in some form and that it is clumped into Diemand's halos. We refer the reader to their favorite introductory cosmology book for a more thorough overview of current research into dark matter.

Perhaps the simplest evidence for DM comes from plotting the velocities of luminous objects in the Galaxy as a function of their distance from the Galactic center, a plot that is frequently referred to as a "Galactic rotation curve". Newtonian gravity predicts that the velocity of an orbiter around the Galactic center scales as

$$v \sim \sqrt{m_{encl}(r)/r}$$

where $m_{encl}(r)$ is the mass enclosed within the orbit and r is the radius from the Galactic center. (See, for example, Eqn. (4) for the case of a comet orbiting the sun.) One therefore expects to see a $\sqrt{1/r}$ dependence in the Galactic rotation curve. However, observations reveal a flat rotation curve beyond a certain radius, which means stars at vastly different distances from the Galactic center orbit

at the same speed. One solution to the disharmony between the predictions of Newtonian gravity and astronomical observations is to propose that luminous matter only makes part of $m_{encl}(r)$ and that the presence of additional "dark" matter is present to make up for the matter deficit suggested by a flat rotation curve.

A variety of proposals for the constituents of DM have been proposed, including WIMPs, superWIMPS, cosmic strings, oodles of super-symmetric particles, small black holes, axions, etc.... The distribution of the cosmic microwave background and the formation rates of galaxies seem to suggest that DM is "cold"; i.e., it was non-relativistic when galaxy formation commenced. One of the currently favored cold DM candidates is the lightest super-symmetric partner, the neutralino. Diemand's simulations favor neutralinos, but it is important to realize that the model we will build in §4 is independent of the particle content of DM; we only care that it can clump into halos.

With these thoughts in mind we now turn to a discussion of the Oort cloud.

3 Properties of the Oort cloud

Fernandez [4] and Weissman [9] have written excellent surveys of what is currently known about the Oort cloud and we refer the interested reader to them for more information than we are able to or need to include. Here we summarize their summaries by only including details that are relevant to this paper.

In 1950 in an attempt to explain the distribution of orbital energies of the 19 long period comets known at the time, the astronomer Jan Oort proposed the existence of a large cloud of comets surrounding the solar system between roughly 10^4AU and 10^5AU that is held in place by the gravitational field of the sun. Much of what is currently known about the Oort cloud comes from studying more than 190 long period comets currently on record (see, e.g., the *Catalogue of Cometary Orbits* by B. Marsden and G. Williams) and by comparing this data with numerous Monte-Carlo simulations. Ordinarily these simulations quantify the effects several sources of perturbation have on the Oort cloud. In particular, many authors have studied whether a given perturbation is strong enough to cause comets to be injected into the solar system or ejected out of the cloud since the fluxes of injected and ejected comets can be inferred from observations.

It is commonly assumed that the comets coalesced out of junk orbiting the sun when the solar system formed 4.5 Gyr ago. The Oort cloud is usually divided into an Inner cloud and an Outer cloud with the transition region chosen to be roughly $2 \cdot 10^4$ AU from the sun where the density of the cloud changes. The Inner cloud has roughly five times more comets than the Outer cloud and may be fed by objects from Uranus-Neptune region. The Outer cloud is more "dynamically active" [9] than the quieter Inner cloud and is seen as the source for comets that enter the solar system. Comets that approach the solar system are sometimes returned to the Oort cloud but frequently they are deflected onto hyperbolic orbits that eject them into space. Inner cloud comets can be gravitationally "pumped" up to replace Outer cloud comets that are lost, but it is likely that around 40% to 60% of the cloud has dissipated over the life of the solar system [9].

Weissman combines the work of several authors to arrive at a population of $\sim 1 \cdot 10^{12}$ comets in the Outer cloud and five times more, or $\sim 5 \cdot 10^{12}$, comets in the Inner cloud for a total population of roughly $6 \cdot 10^{12}$ comets. Weissman estimates a total mass of about 38 earth masses for the entire Oort cloud, meaning each comet is very light.

3.1 Background perturbations to the cloud

Several types of objects are widely recognized to significantly perturb the Oort cloud. Since they interact with each comet regardless of whether any halos exist, we must consider their effects in order to model the environment the Oort cloud sits in as accurately as possible.

The random passage of giant molecular clouds is very disruptive but rare. Ref. [4] cites an estimate of between 1 and 10 encounters over the life of the solar system. These objects are large and heavy, to the tune of $1–2\times10^5 M_\odot$ spread in a sphere of diameter 45 pc. However, we neglect them because they pass by the solar system so rarely and because including them would make it more complex to isolate the effects of Diemand's halos on the cloud.

Stars also randomly pass by (and occasionally through) the cloud. Passing stars randomize the orbits of the comets, can cause a temporary increase in the influx of comets to the solar system, and can strip comets out of the cloud. Weissman estimates that between 5000 and 8000 stars have traveled inside 10^5 AU of the sun and may have ejected roughly 10% of the cloud's population over the life of the solar system. Fernandez estimates that roughly 7 stars per Myr approach within 1 pc of the sun. As we will see in §§4.5.5 & 7.3, halos pass by the solar system more frequently than Fernandez's estimate for stars. We do not wish to confuse the effects of halos (especially heavy ones) with those of passing stars so we do not consider the effects of passing stars.

Many authors have considered the effects of the largest planets, Jupiter and Saturn, on the Oort cloud. The combined mass of all the planets is fraction of 1% of the mass of the sun so one can neglect the planets far from the solar system. However, the effects of the heaviest planets can be very important for comets near the solar system. Jupiter and Saturn serve as a shield for the inner planets by deflecting the orbits of comets back into the Oort cloud or by ejecting comets onto space by accelerating them into hyperbolic orbits. We neglect the effects of all the planets because we are concerned with perturbations that are largely outside the solar system, and because we aren't interested in accurately modeling the dynamics of comets inside the solar system.

The matter near the Galactic plane, called the Galactic disk, is frequently recognized as a strong source of perturbations of the Oort cloud. Since the disk is modeled in the Galactic plane, it cannot affect the component of a comet's angular momentum along the Galactic z-axis, making it difficult for the disk to bring comets into the solar system without the assistance of other perturbing bodies. The disk's effects are greatest for comets whose orbital planes are perpendicular to the disk. We will model the effects of the Galactic disk in §4.4.2.

The bulge at the center of the Galaxy is very massive (roughly $1.3\times10^{11} M_\odot$ according to Ref. [4]) but very far away (roughly 8.5 kc) from the solar system. It probably played a role in randomizing the orbits of comets and may assist in bringing comets into the solar system. Its effects are on average about 5 times less than those of the Galactic disk [4] but in certain configurations the bulge can dominate, such as when a comet's orbital plane is parallel with the disk or when the solar system is far from the disk. We will model the effects of the Galactic bulge in §4.4.3.

Weissman and Fernandez also cite several non-Galactic sources of perturbation to the Oort cloud. Comets occasionally collide, they interact with each other gravitationally, and comets accelerate by jetting material as they approach the hot sun. We will simulate 1000 comets spread over many pc^3 so we do not need to worry about collisions. We also the gravitational interactions of comets with each other because these forces are small compared to the forces due the sources of perturbation we listed above. Jetting is unimportant far from the sun so we neglect it as well.

4 Construction of the model

In this section we will present the assumptions that went into building our model of the Oort cloud and dark matter halos. Then we will determine appropriate initial conditions for the comets and halos. Next we will write equations to describe the interactions of the comets with the sun, the disk, the bulge, and passing halos. Finally, we will use these force equations to write the equations of motion for the comets. We will discuss how we numerically solved them in §5.

4.1 Assumptions

After Ref. [2] we initially confine the Oort cloud to the spherical shell between radii $r_{inner} = 0.1$ pc and $r_{outer} = 0.5$ pc from the sun; however no such restrictions exist once each simulation begins. Over time the perturbations we discussed in the previous section are generally assumed to have randomized the orbits of the comets. We therefore randomly distribute 1000 comets uniformly between these radii and we randomly pick initial velocity vectors in such a way as to produce orbits that are initially stable. This effectively selects the most stable comets in order to determine the maximum effect the halos could have on the Oort cloud.

As we mentioned above, we neglect the interactions of the comets among themselves because such interactions are weak and because computing the interactions of 1000 comets with each other would make our simulations irritatingly slow.

Since we do not consider the comets to be sources of gravitational fields, we do not need to give them masses (m_{comet} appears in several equations, but it drops out of the equations of motion that determine the comets' trajectories).

Diemand indicates that the earliest "dark matter halos to collapse and virialize are smooth triaxial objects of mass $10^{-6}\mathrm{M}_{\odot}$ and half mass radii of 10^{-2} parsecs." The outer boundary of the Oort cloud is 0.5 pc so, if we generate halos with trajectories that don't bring them too close to the solar system, the shell theorem allows us to treat the halos as point objects. This assumption has several advantages. First, our results are independent of speculative models of the substructure of the halos. Second, we increase the effects of the halos on the Oort cloud by assuming all of each halo's mass interacts with each comet instead of only the fraction contained between the comet and the center of the halo. Third, we reduce the complexity of and so increase the speed of our simulations.

We assume for simplicity that all the halos in a given simulation have the same mass.

We create each halo at a random position on the surface of a sphere of radius 3 pc from the sun. This distance is chosen arbitrarily but is large enough that the cloud is insensitive to its exact value for lighter halos but small enough that we don't waste computer time creating and moving halos that will have a negligible effect on the Oort cloud. All the halos are given a velocity component toward the solar system, as we will discuss in §4.5.2. We delete halos once they reach the other side of the sphere containing them to increase the speed of the simulations and to preserve a degree of symmetry by creating and deleting halos at the same distance from the sun. We will discuss how we do this in §6.2.

The comets are very light compared to the halos and the halos move much more quickly as one can see by comparing Figs. 4 with 14, making their collective influences on the halos very small. The halo move fairly rapidly across this space, with a transit time on the order of 11,000 years (see §4.5.5), which does not give the sun, bulge, and disk much time to cause significant deviations from linear trajectories. We consider it a fair approximation to ignore all perturbations to the halos

and assume they move along straight lines. We further assume that the halos do not interact with each other. These assumptions considerably simplify the simulation by reducing the number of interactions that must be modeled.

We will run simulations for 1000 comets, with halo masses ranging from 0.001 $M_\odot$ to 10 $M_\odot$. We will use a time step of 50 years since this time step provides an excellent balance between accuracy and speed. All of our simulations ran for 1 Gyr.

4.2 Coordinate system and other conventions

We use lower case letters to represent the position and velocity coordinates, vectors, angles, and other variables for comets. To make our equations more transparent, we use upper case letters as much as possible to represent similar quantities for all other objects.

We use $\widehat{\mathcal{X}}$, $\widehat{\mathcal{Y}}$, and $\widehat{\mathcal{Z}}$ for the Galacto-centric coordinate system, whose origin is the Galactic bulge, and we use $\hat{x}$, $\hat{y}$, and $\hat{z}$ for helio-centric coordinates. We used helio-centric coordinates for our simulations but will initially write some of the theory that follows in Galacto-centric coordinates.

All the force vectors below are the forces felt *by* each comet due *to* some source(s) of force.

We we write x instead of x_i for the i^{th} comet since we do not need to indicate which comet we're talking about since the comets do not interact with each other and can be considered independent.

We indicate that a quantity is a variable by italicizing it but units are not italicized. In certain equations the sun's mass is italicized since it is a variable (e.g., Eqn. (2) while in others it is not italicized since it is used as a unit (e.g., Eqn. (28).

4.3 Generating the Oort cloud

4.3.1 Picking the comets' initial positions and velocities

In order to test the stability of the Oort cloud under the influences of passing dark matter halos, the Galactic bulge, and the Galactic disk, we first need to generate a stable cloud of comets in order to ensure that any comets that escape the cloud do so because of these perturbations instead of because of the initial conditions we give them.

We found that randomly picking the initial positions and velocities (less than the escape velocity) led to very unstable orbits, with many comets escaping the sun's gravitational pull very quickly. But by carefully choosing the magnitude and direction of each comet's velocity vector, it is possible to generate perfectly stable orbits. It is fairly straight-forward to pick the initial position for a comet, but picking a matching velocity vector turns out to be subtle.

We start by picking the comet's radius from the sun and we will use this to calculate the speed that is necessary for a stable orbit. We will then describe the algorithm we used to randomly orient the position and velocity vectors in three dimensions.

We refer the reader to Fig. 4.3.1 for an illustration (not to scale!) of the variables that appear in the following sentences. We first randomly pick a radius, r, that lies between r_{inner} and r_{outer} (defined in §4.1). Because we want the comet to start at a random position along its orbit, we also randomly pick the two apsidal distances r_1 and r_2 using the following criteria. In order for the orbit to be entirely contained within the Oort cloud, we require that the distance, r_1, between the comet and the focus that is closer to it, f_1, should be between the inner boundary of the Oort cloud and its radius from the sun. Likewise, we require that the distance, r_2, between the comet and the further focus, f_2, should be between the comet's radius from the sun and the outer boundary of the Oort

cloud. In symbols,

$$r_{inner} \leq r_1 \leq r \leq r_2 \leq r_{outer} \tag{1}$$

The equalities correspond to circular orbits where r, r_1, and r_2 satisfy one of

$$\begin{aligned} r_{inner} &= r = r_1 = r_2 \\ r &= r_1 = r_2 \\ r &= r_1 = r_2 = r_{outer} \end{aligned}$$

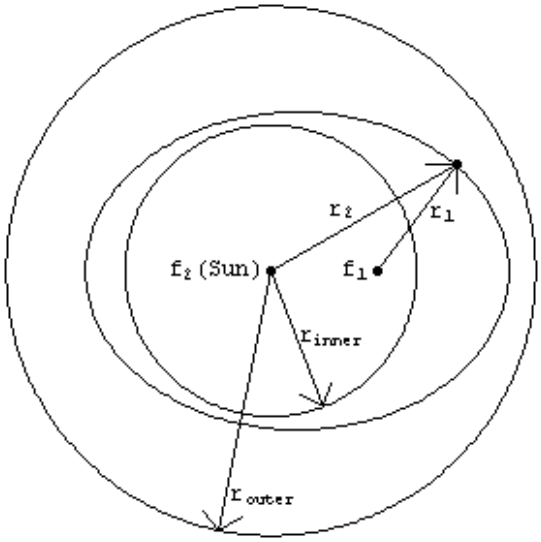

Figure 1: The initial radius of a comet's orbit is chosen between the inner and outer boundaries of the Oort cloud, r_{inner} and r_{outer}. The apsidal distances, r_1 and r_2, are the distances to a comet from the two foci, f_1 and f_2, of its elliptical orbit, with the sun at f_2. We have sketched a case where $r = r_2$ and r_1 meet at the same point to show how the variables are related to each other, but such a coincidence is unlikely since we have randomly chosen r, r_1, and r_2 according to Eqn. (1).

The semi-major axis distance, a, of the ellipse is

$$a = \frac{r_1 + r_2}{2}$$

For an elliptical orbit, a is related to the energy by (see, e.g., Ref. [5])

$$a = \frac{-Gm_{comet}M_{\odot}}{2E} \tag{2}$$

where G is Newton's gravitational constant, where m_{comet} is the mass of the comet, and where E is the energy of the comet. E is the sum of the kinetic and potential energies of the comet

$$E = \frac{1}{2} m_{comet} v^2 - \frac{G m_{comet} M_\odot}{r} \tag{3}$$

Inserting Eqn. (3) into Eqn. (2) and solving for the speed, v, of the comet reveals

$$v = \sqrt{G M_\odot (\frac{2}{r} - \frac{1}{a})} \tag{4}$$

We note that the implied speed that is necessary for a stable orbit only depends on the constants G and $M_\odot$, on the comet's distance from the sun, and on the comet's semi-major axis, which in turn depends on the two apsidal distances we randomly picked. In other words, the condition that a comet have a stable orbit removes our control over its speed once we have chosen r, r_1, and r_2.

We orient the comet's position vector in space by randomly picking its azimuthal and polar angles. It is incorrect to uniformly pick the polar angle θ such that $0 \leq \theta \leq \pi$. Since there is less surface area and volume at the poles than at the equator, picking θ in this manner will put significantly more comets at the poles than along the equator; this is true because dA and dV depend on θ. One way to remedy this problem (see, e.g., Ref. [10]) is to compute randomly pick θ according to

$$\theta = \cos^{-1}(2u - 1) \tag{5}$$

where u is randomly chosen between 0 and 1. Intuitively this is correct since the element of solid angle can be written

$$d\Omega = \sin\theta \, d\theta \, d\phi = -d(\cos\theta) \, d\phi$$

which implies that we ought to pick $\cos\theta$ randomly instead of θ. Of course, the area and volume elements do not depend on ϕ so it is appropriate to randomly pick ϕ in the interval

$$0 \leq \phi \leq 2\pi \tag{6}$$

We now convert r, θ, and ϕ to cartesian coordinates using the usual transformation

$$\begin{aligned} x_1 &= r \sin\theta \cos\phi \\ x_2 &= r \sin\theta \sin\phi \\ x_3 &= r \cos\theta \end{aligned} \tag{7}$$

where, as usual, 1 corresponds to the x component, 2 corresponds to the y component, and 3 corresponds to the z component.

We recall that the position and velocity vectors are perpendicular for an body in uniform circular motion. Even though a comet's orbit in not in general circular, it is not outrageously eccentric either, so we can get away with orienting the position and velocity vectors such that they are orthogonal.

Orthogonal position and velocity vectors satisfy the property

$$\vec{x} \cdot \vec{v} = x_1 v_1 + x_2 v_2 + x_3 v_3 = 0$$

which we solve for v_1, v_2, and v_3 to get

$$\begin{aligned} v_1 &= -(x_2 v_2 + x_3 v_3)/x_1 \\ v_2 &= -(x_1 v_1 + x_3 v_3)/x_2 \\ v_3 &= -(x_1 v_1 + x_2 v_2)/x_3 \end{aligned} \tag{8}$$

The first of these relations suggests that we can randomly pick v_2 and v_3 and use these values in combination with $\vec{x}$ from eqn. (7) to calculate v_1 as long as $v_1^2 + v_2^2 + v_3^2 = v^2$ where v^2 is the square of eqn. (4). Our random number generator outputs numbers between 0 and 1, so we first assign values to v_1, v_2, and v_3 in this range, then we divide by $v_1^2 + v_2^2 + v_3^2$, and finally multiply by v to give the velocity the correct magnitude. This procedure unfairly biases v_1 by forcing it to pick up the "slack" left by randomly picking v_2 and v_3, so we also randomly pick $k = 1$, 2, or 3 and use either the first, second or third of eqns. (8) so that each coordinate receives the same weight. For example, if $k = 2$ we randomly pick v_1 and v_3, we use $\vec{x}$ to compute v_2, then we multiply each component by v and divide each component by $v_1^2 + v_2^2 + v_3^2$ produce the components of the initial velocity vector.

In §7.1 we plot and discuss the distributions of the components of these vectors for the comets we simulated to check that the above considerations were implemented correctly in our code.

4.4 Background forces on the comets

In §3.1 we decided to include the effects of the sun, the Galactic disk, and the Galactic bulge in order to accurately model the Oort cloud. In the sections that follow we will write equations describing the forces felt by a comet due to each of these objects.

4.4.1 The sun

The strongest influence on the motion of each comet (inside Hill's surface) and the reason for its orbit around the solar system is the gravitational attraction the comet feels toward the sun. This attraction is given by

$$\vec{F}_{sun}(t) = -\frac{G m_{comet} M_\odot}{r(t)^3} \vec{r}(t) \tag{9}$$

where we have used Newton's formula for the gravitational force between two objects.

4.4.2 The Galactic disk

The solar system oscillates above and below the Galactic disk according to the equation of motion (Ref. [4])

$$\frac{d^2 \mathcal{Z}}{dt^2} = -4\pi G \rho_{disk} \mathcal{Z}(t) \tag{10}$$

where $\mathcal{Z}(t)$ is the Galacto-centric $\mathcal{Z}$-coordinate of the solar system (since the disk is in the Galactic $\mathcal{XY}$-plane) and where ρ_{disk} is the matter density of the Galactic disk. Ref. [4] combines several experimental results to arrive at

$$\rho_{disk} \approx 0.10 \frac{\mathrm{M}_\odot}{\mathrm{pc}^3}$$

Eqn. (10) gives rise to a vertical oscillation with a half-period of

$$T_{disk,1/2} = \frac{2\pi}{\omega_{disk}} \approx 42 \text{ Myr} \tag{11}$$

where

$$\omega_{disk} = \sqrt{4\pi G \rho_{disk}}$$

during which time the sun reaches a maximum height of approximately 70 pc above the disk.

The interaction between the Galactic disk and an object does not depend on any parameter of the object except its $\mathcal{Z}$ coordinate. Since the disk attracts both the sun and each comet, it is easy to see that the *net* force on a comet compared to the sun depends only on the *difference* in their $\mathcal{Z}$ coordinates, which is just the helio-centric z coordinate of the comet. We modify Ref. [4]'s work by including m_{comet} and $f(t)$ so that the net force due to the disk is

$$\vec{F}_{disk}(t) = -m_{comet}\omega_{disk}^2 f(t) z(t) \hat{z} \tag{12}$$

where $f(t)$ is a square wave that is $+1$ for 42 million years, -1 for 42 million years, and then repeats the cycle to simulate the motion of the solar system above and below the disk.

4.4.3 The Galactic bulge

We can get an upper bound for the effects of the tremendous mass at the center of the Milky Way by assuming that all the mass of the Galaxy resides in a bulge at its center, where

$$M_{bulge} = M_{Galaxy} \approx 1.3 \cdot 10^{11} \mathrm{M}_{\odot}$$

is the value used by Ref. [4]

The sun is rotating around the Galactic center at a distance of roughly

$$R_{sun-bulge} \approx 8.5 \text{ kpc}$$

with a period, T_{bulge}, of roughly 225 million years, giving rise to the associated angular frequency

$$\omega_{bulge} = \frac{2\pi}{T_{bulge}} \approx 2.79 \cdot 10^{-8} \text{ rad/year} \tag{13}$$

The solar system will orbit the Galactic center roughly 4 times during each one billion year simulation and the solar system will also move above and below the bulge as it oscillates above and below the Galactic disk. It is natural to use Galacto-centric coordinates where the solar system moves around the fixed disk and bulge, but we can also leave the sun fixed and move the disk and bulge around the solar system. We choose the latter since it is easier to keep track of the evolution of the comets and halos in helio-centric coordinates because using Galacto-centric coordinates would force us to transform between coordinate systems for operations like creating halos, determining their distance from the sun, binning the comets by radius from the sun, etc... (we will discuss all of these operations later). In short, there is only one disk and one bulge, but there are thousands of comets and halos, so we choose the coordinate system that is more natural for the greatest number of objects.

Since we have chosen to put the sun at the origin, the disk instead moves above and below the solar system. The disk's movement with respect to the sun is governed by an equation similar to Eqn. (10), which gives rise to the solution

$$Z(t) = -A \sin(\omega_{disk} t)$$

where we have put the disk at $Z = 0$ when $t = 0$, where we have made the disk intially move down (or, alternatively, the sun move up), and where $A = 70$ pc is the maximum amplitude of the motion. The bulge and the disk are both in the Galactic $\mathcal{XY}$-plane so we tie the vertical motion of the bulge to the vertical motion of the disk. We also assume that the bulge moves in uniform circular motion at a radius of $R_{sun-bulge}$ at an angular frequency of ω_{bulge}; the bulge's equations of motion are familiar sines and cosines in the sun's xy-plane. The complete equations describing the position of the bulge with respect to the solar system at time t are then

$$\begin{aligned} X_{bulge}(t) &= R_{sun-bulge} \cos(\omega_{bulge} t) \\ Y_{bulge}(t) &= -R_{sun-bulge} \sin(\omega_{bulge} t) \\ Z_{bulge}(t) &= -A \sin(\omega_{disk} t) \end{aligned} \tag{14}$$

where the initial conditions are (1) the bulge sits at $(R_{sun-bulge}, 0, 0)$ at $t = 0$, (2) the bulge moves clockwise about the sun since the sun moves clockwise (as seen from the positive $\mathcal{Z}$ axis) about the Galactic center, and (3) the sun initially moves in the $+\hat{\mathcal{Z}}$ direction so that the bulge and disk initially move in the $-\hat{z}$ direction.

These equations are the components of the vector joining the sun and bulge, which we call $\vec{R}_{bulge}(t)$, whose magnitude

$$|R_{bulge}(t)| = \sqrt{X_{bulge}(t)^2 + Y_{bulge}(t)^2 + Z_{bulge}(t)^2} = \sqrt{R_{sun-bulge}^2 + Z_{bulge}(t)^2}$$

is the distance between the sun and the bulge at time t.

We compute the interaction of the bulge with each comet using Newton's gravitational law, but we need to subtract out the interaction of the bulge with the sun to get the *net* force on each comet with respect to the sun. We do this (as we did for the disk) because the bulge causes the sun to move, which causes the origin in helio-centric coordinates to move.

Putting these ideas together, the net interaction between a comet and the bulge is

$$\vec{F}_{bulge}(t) = -GM_{bulge} m_{comet} \left(\frac{\vec{R}_{bulge}(t) - \vec{r}(t)}{|\vec{R}_{bulge}(t) - \vec{r}(t)|^3} - \frac{\vec{R}_{bulge}(t)}{|\vec{R}_{bulge}(t)|^3} \right) \tag{15}$$

where $\vec{R}_{bulge}(t) - \vec{r}(t)$ is the vector pointing from a comet toward the bulge, where the first term is for the interaction of the bulge with a comet, and the second term subtracts out the movement of the origin due to the effects of the bulge on the sun.

4.5 Dark matter halos

In the next two sections we suppress the index j for the j^{th} halo with the understanding that each halo has its own copy of each of the equations we will write.

4.5.1 Picking the position vectors

We generate all halos on the surface of a sphere at a radius of $R = 3$ pc from the sun and keep them inside this sphere by giving them a giving it a velocity component toward the sun (see the next section) and by deleting halos that stray outside this sphere (see §6.2.1). We pick the position of each halo by randomly choosing its spherical coordinates such that the initial distribution of halos will be uniform across the surface of the sphere containing them. We pick Θ_p and Φ_p using the same distributions we used in §4.3.1 to pick the spherical coordinates for the comets (the subscript p (for position) has been added to distinguish these angles from those appearing in the next section). That is, Θ_p and Φ_p satisfy equations similar to Eqns. (5) and (6), namely

$$\begin{array}{c} \Theta_p = \cos^{-1}(2v - 1) \\ 0 \leq \Phi_p \leq 2\pi \end{array} \tag{16}$$

where v is chosen randomly between 0 and 1.

We then transform to cartesian coordinates via

$$\begin{array}{l} X = R \sin\Theta_p \cos\Phi_p \\ Y = R \sin\Theta_p \sin\Phi_p \\ Z = R \cos\Theta_p \end{array} \tag{17}$$

to obtain the components of the initial position vector of each halo.

4.5.2 Picking the velocity vectors

We set $V_{coord} = 150$ km/s for the maximum velocity for each component of the velocity vector, giving rise to a maximum velocity of

$$V_{max} = \sqrt{3} V_{coord} \approx 260 \text{ km/s} \tag{18}$$

As we will see in §4.5.5, a certain number of halos must be present inside the 3 pc sphere containing them to maintain the density of dark matter near the solar system. Also, we do not wish to waste computer time generating halos that will immediately move outside this sphere where they will be deleted. We therefore bias the velocity vector of each halo with a negative component in the $\hat{r}$ direction so that each halo will tend to progress toward the solar system. The polar and azimuthal velocities, however, can be either positive or negative with magnitudes between 0 and V_{max}. In symbols, the components of the velocity vector are randomly chosen from the intervals

$$\begin{array}{l} -V_{coord} \leq V_R \leq 0 \\ -V_{coord} \leq V_\Theta \leq V_{coord} \\ -V_{coord} \leq V_\Phi \leq V_{coord} \end{array}$$

We then orient the velocity vector in space by randomly picking Θ_v and Φ_v using Eqns. (16) where $p \rightarrow v$ (the subscript v stands for "velocity" to distinguish the angles from those appearing in the previous section). We now pass from spherical to cartesian coordinates by writing

$$\vec{V} = V_R \hat{r} + V_\Theta \hat{\theta} + V_\Phi \hat{\phi} \tag{19}$$

and then converting the spherical unit vectors to cartesian unit vectors through the relations

$$\begin{aligned}\hat{r} &= \sin\Theta_v\cos\Phi_v\hat{x} + \sin\Theta_v\sin\Phi_v\hat{y} + \cos\Theta_v\hat{z}\\ \hat{\theta} &= \cos\Theta_v\cos\Phi_v\hat{x} + \cos\Theta_v\sin\Phi_v\hat{y} - \sin\Theta_v\hat{z}\\ \hat{\phi} &= -\sin\Phi_v\hat{x} + \cos\Phi_v\hat{y}\end{aligned} \tag{20}$$

By substituting Eqns. (20) into Eqn.(19), we obtain

$$\begin{aligned}V_X &= V_R\sin\Theta_v\cos\Phi_v + V_\Theta\cos\Theta_v\cos\Phi_v - V_\Phi\sin\Phi_v\\ V_Y &= V_R\sin\Theta_v\sin\Phi_v + V_\Theta\cos\Theta_v\sin\Phi_v + V_\Phi\cos\Phi_v\\ V_Z &= V_R\cos\Theta_v - V_\Theta\sin\Theta_v\end{aligned} \tag{21}$$

for the components of the velocity vector of each halo.

In §7.2 we will discuss and plot histograms of the initial positions and velocities of the halos.

4.5.3 Equations of motion

As we discussed in §4.1, we assume the halos do not interact. This means each halo's velocity vector is constant so each halo moves along a line. The resulting equations of motion are, therefore,

$$\begin{aligned}X(t_{n+1}) &= X(t_n) + V_X h\\ Y(t_{n+1}) &= Y(t_n) + V_Y h\\ Z(t_{n+1}) &= Z(t_n) + V_Z h\end{aligned} \tag{22}$$

where $X(t_n)$, $Y(t_n)$, $Z(t_n)$, are the components of the halo's position vector at time n, where V_X, V_Y, and V_Z are the components of the halo's constant velocity vector, Eqns. (21), and where h is the time between the adjacent time steps t_n and t_{n+1}.

4.5.4 Impact parameters

The assumption that the halos move along straight lines allows us to readily compute their impact parameters with respect to the sun using each halo's initial position and velocity vectors. This is handy since it means one does not have to waste time finding the point of closest approach numerically. We refer the reader to the sketch below for a graphical representation of the relationships among the variables that follow. In this section we restore the index i for the i^{th} halo for reasons that will become clear momentarily.

We first calculate

$$\sin\beta_i = \frac{|\vec{X}_i \times \vec{V}_i|}{|\vec{X}_i||\vec{V}_i|}$$

and use this with

$$B_i = |\vec{X}_i|\sin\beta_i \tag{23}$$

to write

$$B_i = \frac{|\vec{X}_i \times \vec{V}_i|}{|\vec{V}_i|}$$

which relates the impact parameter to the initial position and velocity vectors of the i^{th} halo.

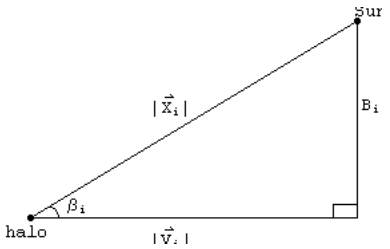

Figure 2: Impact parameter B_i for the i^{th} halo, shown with $|\vec{X}_i|$, $|\vec{V}_i|$, and β_i, which are the magnitudes of the halo's *initial* position and (constant) velocity vectors as given by Eqns. (17) and (21), and the angle between them. We note that $|\vec{X}_i| = R = 3$ pc and that $|\vec{V}_i|$ does not span the bottom leg of the triangle as it appears to in our sketch.

To check that what we have done up to this point is correct, we next derive an expression for the distribution of the impact parameters of many halos. We start from the assumption that the angle β_i between the initial position and velocity vectors of the i^{th} halo should be random between 0 and $\pi/2$ (β_i cannot exceed $\pi/2$ since we designed the velocity vector to have a negative component in the $\hat{r}$ direction); i.e., a halo should be able to point in any direction (as long as it points toward the solar system) with equal probability. We drop the subscript i in the following discussion since we now wish to think of β and B as random variables instead of as discrete quantities containing values for the i^{th} halo.

We already know how to change variables from β to B through Eqn. (23) but we need to determine how the density of states changes under $\beta \to B$ in order to convert the uniform probability distribution from a function of β to a function of B. To do this, we find the cumulative probability distribution as a function of β, convert it to its form over B, and take its derivative with respect to B to get the probability density function over B. This procedure is called the method of distribution functions by mathematicians; see Ref. [8] for more information about this method.

We solve Eqn. (23) for β to get

$$\beta = \sin^{-1}(B/R)$$

where $|\vec{X}_i| = R$ has been made explicit. We now choose an arbitrary impact parameter b with an associated β_b and sum up the number of impact parameters B with an associated β_B between 0 and b. Since $0 < B < b$, we get $0 < \beta_B < \beta_b = \sin^{-1}(b/R)$. The cumulative probability distribution of picking B such that $B < b$ is given by integrating $B(\beta)$ over these angles to obtain

$$\begin{aligned} F(b) &= \textstyle\int_0^{\sin^{-1}(b/R)} R\sin\beta \, d\beta \\ &= -R(\cos(\sin^{-1}(b/R)) - 1) \\ &= -R(\tfrac{1}{R}\sqrt{R^2 - b^2} - 1) \\ &= -\sqrt{R^2 - b^2} + R \end{aligned}$$

We now differentiate $F(b)$ with respect to b and make the replacement $b \to B$ to get

$$f(B) = \frac{d\,F(B)}{dB} = \frac{B/R}{\sqrt{1-(B/R)^2}} \tag{24}$$

for the distribution of impact parameters.

In Fig. 13 we have plotted the distribution of impact parameters for the 50122 halos generated when each has a mass of 10 $M_\odot$ along with the function

$$g(B) = \alpha f(B) = \alpha \frac{B/R}{\sqrt{1-(B/R)^2}} \tag{25}$$

where the constant $\alpha \approx 212.4$ normalizes Eqn. (24) to the data. The excellent agreement between the expected and actual distributions boosts our confidence that the initial conditions for the halos are correct both on paper and in our computer code.

4.5.5 Time interval between halos

As noted in §1, we proceed by computing how the Oort cloud is perturbed by passing dark matter halos, with each simulation testing a different mass for the halos. However, the total amount of dark matter in the neighborhood of the solar system is, on average, constant, even though the amount is only roughly known. In this paper we use

$$\rho_{DM} = 0.05\,\frac{M_\odot}{\mathrm{pc}^3} \tag{26}$$

from Ref. [1]. Since we can choose the masses of the halos but not the total amount of dark matter, we must determine how often we should generate halos of a given mass so that the time average of the total mass of the halos is consistent with Eqn. (26).

All halos are generated and confined inside a sphere of radius 3 pc from the sun, so the volume available to the halos is

$$V = \frac{4}{3}\pi(3\text{ pc})^3 \approx 113.1\,\mathrm{pc}^3 \tag{27}$$

This implies that the average mass of dark matter enclosed in V at any given time is

$$M_{encl} = \rho_{DM} V \approx 5.65\ M_\odot \tag{28}$$

The distribution of halo impact parameters (see Eqn. (25) or Fig. (13)) is highly skewed, so the mean is not as a good a measure of the impact parameter of a typical halo as the median is; the median is 2.5981 pc. The impact parameter and radius of the 3 pc sphere form a right triangle with the third side being the distance between the starting point of the halo and the point of closest approach to the sun. This triangle can be reflected across the line connecting the sun to the point of closest approach, so that twice the length of the third leg of the triangle equals the total distance a typical halo travels to cross the 3 pc sphere. This distance is

$$2\sqrt{(3\text{ pc})^2 - (2.59\text{ pc})^2}$$

where we have used 2.59 pc instead of 2.5981 pc from our data. We will justify this decision in §7.3. To find the minimum time it will take to travel this distance, we divide the distance by an estimate of the halo's top speed, namely Eqn. (18). This means the minimum travel time is approximately

$$t = \frac{2\sqrt{(3 \text{ pc})^2 - (2.59 \text{ pc})^2}}{\sqrt{3}(150 \text{ km/s})} \approx 11403 \text{ years} \tag{29}$$

The time average of the total halo mass in V is given by

$$M_{avg} = \frac{M_{halo}t + 0 * t'}{t + t'} \tag{30}$$

where M_{halo} is the mass of the halo under consideration and where t' is the amount of time that elapses between when a halo leaves the 3pc sphere and when the next halo enters. In other words, t' is the amount of time when the mass enclosed is 0 and t is the amount of time when the mass is M_{halo}. Setting

$$M_{avg} = M_{encl}$$

and solving for $(t + t')$, the total time between halos, we find

$$(t + t') = \frac{M_{halo}}{M_{encl}}t \tag{31}$$

for the time interval between halos so that Eqn. (28) is maintained.

In §7.3 we provide tables and graphs showing the time interval between halos, the average number of halos at any given time, and the total mass density of the halos as a function of the mass we choose for each halo.

We have effectively broken the Galactic dark matter halo entirely into sub-halos in order to maximize the effects the halos could have on the Oort cloud.

4.5.6 Force exerted on each comet

The interaction between halo i and each comet can be expressed using Newton's gravitational equation so that

$$\vec{F}_{halo,i}(t) = -\frac{Gm_{comet}M_{halo}}{|\vec{d_i}(t)|^3}\vec{d_i}(t) \tag{32}$$

where

$$\vec{d_i}(t) = \vec{R}_i(t) - \vec{r}(t)$$

is the vector pointing to the halo from the comet, where $\vec{r}(t)$ and $\vec{R}_i(t)$ are the position vectors of the comet and i^{th} halo. We can expand the denominator of this equation as

$$|\vec{d_i}(t)|^3 = (R_i(t)^2 + r(t)^2 - 2\vec{R}_i(t) \cdot \vec{r}(t))^{\frac{3}{2}} \tag{33}$$

which makes the evaluation of the denominator more efficient for each comet-halo pair since $R_i(t)^2$ and $r(t)^2$ are already computed at each time step; the reasons we keep track of the squared radii will become clear later.

4.6 Summary of all forces on each comet

Here we collect each of the above forces felt by each comet by summing Eqns. (9), (15), (12), and (32).

$$\begin{aligned}
\vec{F}(t,\vec{r}(t)) &= \vec{F}_{sun}(t,\vec{r}(t)) + \vec{F}_{bulge}(t,\vec{r}(t)) + \vec{F}_{disk}(t,\vec{r}(t)) + \vec{F}_{halo}(t,\vec{r}(t)) \\
&= -m_{comet}\left[\begin{array}{c} \frac{GM_\odot \vec{r}(t)}{r(t)^3} + GM_{bulge}\left(\frac{\vec{R}_{bulge}(t)-\vec{r}(t)}{|\vec{R}_{bulge}(t)-\vec{r}(t)|^3} - \frac{\vec{R}_{bulge}(t)}{|\vec{R}_{bulge}(t)|^3}\right) \\ + \omega_{disk}^2 f(t) z(t)\hat{z} + GM_{halo}\sum_i \frac{\vec{d}_i(t)}{|\vec{d}_i(t)|^2} \end{array}\right] \\
&= -m_{comet}\left\{\begin{array}{l} \hat{x}\left[\frac{GM_\odot}{|\vec{r}(t)|^3}x(t) + GM_{bulge}\left(\frac{X_{bulge}(t)-x(t)}{|\vec{R}_{bulge}(t)-\vec{r}(t)|^3} - \frac{\vec{R}_{bulge}(t)}{|\vec{R}_{bulge}(t)|^3}\right)\right. \\ \qquad\qquad \left. + GM_{halo}\sum_i \frac{X_i(t)-x(t)}{|\vec{d}_i(t)|^3}\right] \\ +\hat{y}\left[\frac{GM_\odot}{|\vec{r}(t)|^3}y(t) + GM_{bulge}\left(\frac{Y_{bulge}(t)-y(t)}{|\vec{R}_{bulge}(t)-\vec{r}(t)|^3} - \frac{\vec{R}_{bulge}(t)}{|\vec{R}_{bulge}(t)|^3}\right)\right. \\ \qquad\qquad \left. + GM_{halo}\sum_i \frac{Y_i(t)-y(t)}{|\vec{d}_i(t)|^3}\right] \\ +\hat{z}\left[\frac{GM_\odot}{|\vec{r}(t)|^3}x(t) + GM_{bulge}\left(\frac{Z_{bulge}(t)-z(t)}{|\vec{R}_{bulge}(t)-\vec{r}(t)|^3} - \frac{\vec{R}_{bulge}(t)}{|\vec{R}_{bulge}(t)|^3}\right)\right. \\ \qquad\qquad \left. + GM_{halo}\sum_i \frac{Z_i(t)-z(t)}{|\vec{d}_i(t)|^3} + \omega_{disk}^2 f(t) z(t)\right] \end{array}\right\}
\end{aligned} \tag{34}$$

where m_{comet} has been factored out and where we have summed over all the halos. We have already given formulas for the time dependence of the coordinates of all objects except the comets, so $\vec{F}$ truly is a function of t and $\vec{r}(t)$ since these are only independent variables.

4.6.1 Comets' equations of motion

The equations of motion for each comet are given by solving Newton's second law for each comet, namely

$$\vec{F}(t,\vec{r}(t)) = m_{comet}\frac{d^2\vec{r}}{dt^2}$$

This implies that the time evolution for the coordinates of each comet can be obtained by solving

$$\begin{aligned}
m_{comet}\tfrac{d^2x}{dt^2} &= F_x(t,x(t),y(t),z(t)) \\
m_{comet}\tfrac{d^2y}{dt^2} &= F_y(t,x(t),y(t),z(t)) \\
m_{comet}\tfrac{d^2z}{dt^2} &= F_z(t,x(t),y(t),z(t))
\end{aligned} \tag{35}$$

where F_x, F_y, and F_z are the x, y, and z components of Eqn. (34) and where we have broken down the dependence on $\vec{r}(t)$ into its components. We note that m_{comet} cancels out of Eqns. (34) and (35) as promised, so we no longer write m_{comet} (even though it does appear on one side of some of the equations that follow, it will cancel since those equations are related to Eqns. (34)).

These equations form a second order system of ODEs which we choose to solve using a Runge-Kutta integrator as we will describe in the next section.

5 Solution of the model

5.1 Converting to a system of first order ODEs

The assumption that the comets do not interact with each other has allowed us to write one set of Eqns. (35) for each comet. These sets are completely decoupled in the sense that $\vec{r}_k(t)$ and $\vec{r}_l(t)$ do not talk to each other for the k^{th} and l^{th} comets. This means we can build a single integrator with the same form for each comet.

We note that the first derivative of the comet's position, its velocity, never appears on the right hand side of Eqns. (35), which means they form a conservative second order system of ODEs. We can therefore make them more amenable to Runge-Kutta integration by making the usual substitution

$$\begin{aligned} v_x(t) &= \tfrac{dx}{dt} \\ v_y(t) &= \tfrac{dy}{dt} \\ v_z(t) &= \tfrac{dz}{dt} \end{aligned} \tag{36}$$

so that Eqns.(35) can be written as

$$\begin{aligned} \tfrac{dx}{dt} &= v_x(t) \qquad & \tfrac{dv_x}{dt} &= F_x(t, x(t), y(t), z(t)) \\ \tfrac{dy}{dt} &= v_y(t) \qquad & \tfrac{dv_y}{dt} &= F_y(t, x(t), y(t), z(t)) \\ \tfrac{dz}{dt} &= v_z(t) \qquad & \tfrac{dv_z}{dt} &= F_z(t, x(t), y(t), z(t)) \end{aligned} \tag{37}$$

giving a system of six *first* order ODEs to integrate per comet per time step.

5.2 Runge-Kutta integrator

5.2.1 Numerically advancing the velocities

To numerically integrate the right set of Eqns. (37) we employ a fourth order Runge-Kutta integrator for systems of ODEs (see Ref. [6]). Modified for Eqns. (37), the integrator is

$$\vec{v}(t_{n+1}) = \vec{v}(t_n) + \frac{1}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) \tag{38}$$

where the four trial functions are

$$\begin{aligned} \vec{k}_1 &= hF_x(t_n, \vec{r}(t_n)) \\ \vec{k}_2 &= hF_x(t_n + \tfrac{h}{2}, \vec{r}(t_n) + \tfrac{\vec{k}_1}{2}) \\ \vec{k}_3 &= hF_x(t_n + \tfrac{h}{2}, \vec{r}(t_n) + \tfrac{\vec{k}_2}{2}) \\ \vec{k}_4 &= hF_x(t_n + h, \vec{r}(t_n) + \vec{k}_3) \end{aligned} \tag{39}$$

where h is the time between the consecutive time steps t_n and t_{n+1} and where the vector symbol stands for x, y, and z.

Because evaluating k_{2x} requires a knowledge of k_{1x} (similarly for $x \rightarrow y$ and $x \rightarrow z$) we must first evaluate $\vec{k}_1$ before we can evaluate $\vec{k}_2$. Likewise, we must evaluate each component of $\vec{k}_2$ before $\vec{k}_3$ and $\vec{k}_3$ before $\vec{k}_4$. Since F_x, F_y, and F_z do not depend on each other at each time step, we can evaluate k_{1x}, k_{1y}, and k_{1z} simultaneously, k_{2x}, k_{2y}, and k_{2z} simultaneously, etc.... Evaluating these components simultaneously allows us reuse a number of the calculations that go into evaluating Eqns. (34) for each component of $\vec{F}$. For example, the denominators in Eqns. (34) are the same for the x, y, and z components of the force.

These relationships among the variables mean that the order of operations should be

Table 1: Order of operations

1. Evaluate $k_{1x},\ k_{1y},\ k_{1z}$.
2. Evaluate $k_{2x},\ k_{2y},\ k_{2z}$.
3. Evaluate $k_{3x},\ k_{3y},\ k_{3z}$.
4. Evaluate $k_{4x},\ k_{4y},\ k_{4z}$.
5. Evaluate
   (a) $v_x(t_{n+1}) = v_x(t_n) + \frac{1}{6}(k_{1x} + 2k_{2x} + 2k_{3x} + k_{4x})$
   (b) $v_y(t_{n+1}) = v_y(t_n) + \frac{1}{6}(k_{1y} + 2k_{2y} + 2k_{3y} + k_{4y})$
   (c) $v_z(t_{n+1}) = v_y(t_n) + \frac{1}{6}(k_{1z} + 2k_{2z} + 2k_{3z} + k_{4z})$

5.3 Numerically advancing the positions

After we integrate F_x, F_y, and F_z, to find v_x, v_y, and v_z for the n^{th} time step, the velocities will be just numbers. We can integrate the left half of Eqns. (37) to get

$$\begin{aligned} x(t_{n+1}) &= x(t_n) + v_x \times (t_{n+1} - t_n) = x(t_n) + v_x h \\ y(t_{n+1}) &= y(t_n) + v_y \times (t_{n+1} - t_n) = y(t_n) + v_y h \\ z(t_{n+1}) &= z(t_n) + v_z \times (t_{n+1} - t_n) = z(t_n) + v_z h \end{aligned} \tag{40}$$

where effectively all we have done is to reverse the substitution we made in Eqns. (36) with the assumption that the speeds can be treated as constants over the time h. Since we integrated left half of Eqns. (37) in closed form, we have reduced the number of numerical integrations required to solve Eqns. (37) from six to just three.

5.4 A more efficient order of operations

We can make the numerical evaluation of Eqns. (39) in Table 5.2.1 a great deal more efficient by evaluating several of the terms that are the same for all comets in Eqns. (34) at $t = t_n$, $t = t_n + h/2$ and $t = t_n + h$ only once.

The bulge is at the same position for all the comets at each time step so it makes sense to evaluate

$$\vec{R}_{bulge}(t) = X_{bulge}(t)\hat{x} + Y_{bulge}(t)\hat{y} + Z_{bulge}(t)\hat{z}$$

and

$$-GM_{bulge}\frac{\vec{R}_{bulge}}{|\vec{R}_{bulge}|^3}$$

once at the required times and to make the results available to all the comets.

Likewise, the disk is at the same position for each comet at each time step, so we evaluate and store the term $\omega_{disk}^2 f(t)$ instead of recalculating it for each comet-disk pair.

Evaluating Eqns. (39) also requires a knowledge of the positions of all the dark matter halos, so we use their equations of motion to evaluate and store these positions along with $|\vec{R}_i(t)|^2$ for each i at the required times. Since $t_n + h = t_{n+1}$, we do not need to reevaluate the positions of the halos at t_{n+1} when we advance them in time; instead we can instead copy their position vectors from $t_n + h$.

All told, a more efficient order of operations is given in Table 2 below.

Table 2: A more efficient order of operations

1. Calculate & store the position of the Galactic bulge and its force on the sun.
2. Determine whether the solar system is above or below the Galactic disk and partially calculate its force on the comets.
3. Calculate & store the position of each halo.
4. Evaluate the steps in Table (5.2.1) for x, y, and z where the right hand sides of Eqns. (39) have been partially evaluated in steps 1-3 above.

6 Computational methods

In the sections that follow we present a brief overview of the C++ program we used to carry out the procedures we outlined in §5. Our code appears in Appendix A.

First we will describe how we organized the data and the functions that act on that data into classes, including the input/output and data analysis methods. We will then present the *main*() function that controls the flow of the program. Finally we will discuss how we parallelized part of the program for faster execution.

We will use italics for variables and functions in this section to separate them from our prose; we have put parentheses after the names of functions to indicate that they are functions. For example, we referred to the main function in our code as "*main*()" in the previous paragraph.

6.1 The comet class

The comet class contains variables for a single comet's position and velocity vectors (stored as arrays) and variables for its radius from the sun, its radius squared, its impact parameter squared, and its impact parameter. These values are initialized by the constructor using the methods discussed in §(4.3). We prefer to work with squared distances instead of the distances themselves because finding the distance between two points requires a very slow square root operation.

We compute each comet's radius from the sun every 20,000 years when we bin all the comets by radii (see §6.3.1), otherwise we use the radii squared and the components of the position vectors to compute the terms in Eqns. (34). We use the squared impact parameters until the end of each simulation. At that time $main()$ calls the $calcImpactParameter()$ function to calculate the square root of the squared impact parameter. Then $main()$ directs a function to analyze the distribution of impact parameters (see §6.3.2).

Static functions and variables in C++ have the advantage that only one copy of each is made for all instantiations of a class. This arrangement is ideal for computing and storing the terms discussed in §5.4 for the disk and bulge. The $main()$ function calls the static function $setBulgeDiskForce()$ to evaluate these terms and to store the results in different static variables for the three times $t = t_n$, $t = t_n + h/2$, and $t = t_n + h$.

Similarly, each comet needs to know the position of each halo at each of these times. We make this information available to the comets by writing functions for the halos that compute and store their positions; we will discuss these functions in the next section.

Since we store quantities for the disk, bulge, and halos at $t = t_n$, $t = t_n + h/2$, and $t = t_n + h$, we have written three specialized, private force functions to evaluate Eqns. (34) using the variables appropriate to the each time. This also allows us to modify the position vector of each comet via Eqns. (39) before evaluating the next force function.

The comet class contains a Runge-Kutta integrator, so that each comet has its own integrator with access to the data for the comet to which it is assigned. Each integrator is called by the $main()$ function and carries out the steps in Table 5.2.1 for its comet to advance it from $t = t_n$ to $t = t_{n+1}$.

We achieve another increase in efficiency by shamelessly creating static and private temporary variables; some are global to the class and are used to pass information between functions while others are local to certain functions. It is helpful to use such variables since since we avoid the expense associated with passing variables, passing pointers, allocating memory, and deallocating memory each time frequently used functions are called. This is a shady practice that might keep a computer scientist up at night, but since we are the only one writing and using our code, we consider it safe.

As we have said, the comet class contains sufficient information for a single comet. We create a separate object for each comet and organize all these objects together into an array of comets. By passing a pointer to this array to the output class, we can give it access to the data for all the comets.

6.2 Dark matter halo data structure

In §6.2.2 we will discuss how we organize the data and the functions that act on that data into a halo class. However, it is convenient to first discuss how we collect the halos together using a linked list.

6.2.1 The halo linked list

We use a linked list because halos are frequently added and removed. It is advantageous to use a doubly linked list in which each element contains pointers to the elements both before and after it since the extra pointer to the previous element simplifies scanning the squared radii of each halo, removing it if necessary, and continuing the scan.

The doubly linked list contains *add*() and *remove*() functions to add and remove halos. We added one step to these functions to keep track of how many halos exist at any given time, otherwise the functions are fairly standard and do not require further clarification.

The *deltaPositions*() function cycles through the list (using a standard technique) and calls the *deltaPositions*() function for each halo, which instructs each halo to compute and store its positions at $t = t_n$, $t = t_n + h/2$, and $t = t_n + h$ using each halo's equations of motion. After all the comets' orbits have been integrated from t_n to t_{n+1}, *main*() calls the *advance*() function to cycle through the list of halos calling each halo's *advance*() function to move it from its position at $t = t_n$ to its position at $t = t_{n+1}$. Since $t_n + h = t_{n+1}$, each halo's *advance*() function just copies the halo's position vector at $t_n + h$ to its position vector at t_{n+1} to avoid recalculating it.

Every 500 years, *main*() calls the *radiusCheck*() function to cycle through the halos and remove ones that have strayed too far from the sun using the *remove*() function.

In order to create a data file with the initial conditions of the halos, we ran a simulation that just added halos at the appropriate intervals. Then *main*() called the *initialData*() function in the linked list to print the initial conditions of all the halos to a file for later analysis (this is the data we will present in §7.2). (It took about 1 second to do this since all the data was stored in RAM before writing it to the hard disk. Our previous approach was to write each new halo's initial conditions to the hard disk before adding the next halo. This process took tens of minutes.) This routine also prints data for each halo that is only compiled into the program when this routine is called; otherwise we comment out the lines of code that compute, for example, the impact parameter of the halo.

We provided the computer with a constant seed for the random number generator so that each simulation used the same comets and halos. This means we only had to record the initial conditions once since they were the same for every simulation.

The class also contains several static variables that are created one time in its member functions to avoid the expense of creating and deleting variables for frequently called functions.

By passing a pointer to the head of the linked list to a static variable in the comet class, we give the comets access to the positions of the halos.

6.2.2 The halo class

We store the position and velocity vectors as three dimensional arrays. When a new halo is created by the *add*() function in the linked list, it calls the halo's constructor. This function initializes the halo's position and velocity vector using the methods discussed in §§4.5.1 & 4.5.2.

We never need the distance of the halos from the sun since we can instead use the distance squared. This allows us to avoid calculating many costly square roots.

The *deltaPositions*() and *advance*() were described in the last section so we do not need to write more about them.

To form the linked list, each halo contains pointers to the halo before it and the halo after it, but these pointers are controlled by functions in the linked list class.

6.3 The output class

We have bundled most of the input/output routines into the output class along with the functions that analyze the orbits of the comets.

When the output class is instantiated by the *main*() function, the class's constructor creates a README file to output the parameters of the simulation that is being run.

After all the comets have been created, the *cometInitialGridData*() can be called by *main*() to write the initial conditions for each comet to a file. This data appears in §7.1.

In §6.4.1 we will discuss the two types of parallelization we used to speed up the execution of our simulations. One of these types required us to write the initial conditions for the comet to a file, but this file does not need to contain all the data outputted by the *cometInitialGridData*() function. Instead, the *cometGridParallelData*() function can be called by *main*() to write the initial positions and velocities (but no other information) to a file for this purpose. The data can then be read from a file by the *populateGrid*() routine.

We keep track of how far each comet is from the sun and report the distribution of the comets' radii using the bin and impact parameters routines, which we will discuss in §§6.3.1 & 6.3.2.

6.3.1 Binning routine

We bin the comets every 20,000 years. We chose this time so that it is small enough to capture the movements of comets but long enough that we do not generate excessive amounts of data.

We bin the comets into radial bins of width 0.05 pc with the first bin at $r = 0$ and the last bin at $r = 1$ pc for a total of 19 bins. Initially the Oort cloud was confined to radii between 0.1pc and 0.5pc but the bins cast a larger net over the Oort cloud to allow for some spreading over time.

We also sum up the number of comets that are within 1 pc of the sun and subtract this number from the total number of comets to get the number of escaped comets. We consider the former number the population of the Oort cloud and make plots of it as a function of time in §7.4. (In this paper we only plot the population as a function of time and do not plot the movement of the comets among the bins.)

In order to reduce the number of (slow) hard disk accesses, we create a *bins* class to store many sets of binned data in RAM before writing it to disk. For faster simulations the savings in time is significant since the data needs to be binned often; for these simulations use a large number of *bins*. But for slower simulations the percent of the computer time spent writing data to the disk is small, so we reduce the number of *bins* so the size of the file of binned data more accurately reflects the progress of the simulation.

6.3.2 Comet's impact parameters

We use the each comet's initial distance from the sun to set its initial squared impact parameter. After updating its comet's position and velocity, each Runge-Kutta integrator checks whether the new distance squared from the sun is less than the squared impact parameter. If the new distance squared is smaller than the squared impact parameter, the latter is updated with the former. In this way each simulation determines the distance of closest approach to the sun for each comet.

The *cometImpactParameters*() function loads the impact parameters for all the comets and counts how many are smaller than 40 AU (approximately the radius of Pluto's orbit), 100 AU, and 1000 AU. The impact parameters are then binned using the same techniques we described in

the previous section (although these plots do not appear in this paper since there are not many interesting conclusions to be drawn from them). We will plot and discuss this data in §7.5.

6.4 Program flow: the *main*() function

Most of the physical constants we use are stored as global constants before *main*() is called.

When *main*() is called, it first seeds the random number generator with 100 as the numerical seed. Picking a constant seed allows us to run simulations on multiple machines with the same initial conditions for the comets and halos. This allows us to reliably compare the results of simulations that differ by a single parameter such as the mass of the halos. We note, however, that since there are more lighter halos than heavier ones, only the first group of halos is shared among different simulations.

Next *main*() creates an array of comets, creates the linked list of halos, and instantiates the output class. When the array of comets is initialized, the comets' constructors set each comet's initial conditions. However there are no halos until the simulation starts. The output class now receives a pointer to the array of comets and the comet class receives a pointer to the head node of the halo linked list.

A global function, *setHaloInterval*(), is then called to calculate the time interval between the halos using the methods described in §4.5.5.

At this point we are ready to begin the simulation. Since we wish to bin the comet's radii, integrate the comets' orbits, add halos, and check the squared radii of the halos all at different times, we use the following scheme to control the flow of the simulation. We store when data should be binned, when the comets' orbits should be integrated, when the next halo should be added, and when the halos' radii should be checked in the variables *nextBinTime*, *nextAdvance*, *nextAddTime*, and *nextHaloCheck*, respectively. The following series of if-statements are wrapped by a for-loop that starts at $t = 0$, terminates at $t = 1$ Gyr, and proceeds in increments of one year.

If $time = nextBinTime$, *main*() calls the *binCometData*() function in the output class. If $time = nextAddTime$, *main*() calls the *add*() function in the linked list to add another halo. If $time = nextHaloCheck$, *main*() calls the *radiusCheck*() function in the linked list to remove halos that have strayed too far from the sun. If any halos were removed, *main*() gives a static variable in the comet class the position of new head of the linked list in case the halo stored in the old head node was deleted. If $time = nextAdvance$, *main*() calls *setBulgeDiskForce*() in the comet class and *deltaPositions*() in the linked list. Then *main*() uses a for-loop to cycle through the array of comets, calling each comet's Runge-Kutta integrator in sequence. Once all the comets have been advanced from $t = t_n$ to $t = t_{n+1}$, *main*() calls the *advance*() function in the linked list. Of course, if one of the if-statements leads to the execution of any of the above instructions, the corresponding *next* variable is incremented to its next value; e.g., *nextBinTime* is increased by 20,000 years when *binCometData*() is called.

After the simulation ends, *main*() calls the *calcImpactParameter*() function for each comet and calls the *cometImpactParameter*() routine in the output class.

6.4.1 Parallelization

As we will find in §7.3, there are on average about 5800 halos at any given time when their mass is 0.001 $M_\odot$. We must compute the interaction of each halo with each of the 1000 comets for each of x, y, and z (see Table (5.2.1). Fortunately we can reuse the square roots in the denominators of

Eqns. (34) for the three coordinates, but we must still evaluate roughly $4 \times 1000 \times 5800 = 23,200,000$ square roots for each comet-halo pair. We must also evaluate four square roots per comet to compute the interaction of each comet with the bulge. All of these square roots are necessary to evaluate Eqns. (34) for each comet's integrator at each 50 year time step; there are 20,000,000 steps in our 1 Gyr simulations.

Computing these interactions is by far the slowest part of our program but, since the comets are independent, the Runge-Kutta integrators do not depend on each other during each time step. This means the integrators can be executed in sequence on a single CPU, or they can be put on separate threads on a multiple processor machine or cluster. We use the OpenMP directives to split the Runge-Kutta integrators into threads using the fork/join model of parallelization. Once all the threads (i.e., integrators) have completed, the program returns to serial execution. Operations like creating and deleting halos, advancing the halos in time, computing the positions of the disk and bulge, etc... cannot be parallelized. While these operations are being executed in serial, some CPUs have to wait idly before another thread (i.e., integrator) can be handed to them. Even though this approach does not make full use of each CPU, it did greatly increase the speed of our simulations.

The second approach to parallelizing our simulations is the following. In §6.3 we mentioned that we recorded the initial conditions of the comets to a file for later use. We wrote Linux scripts to parse this data into chunks so that separate simulations could be run on different comets. (We called the input files for each of these chunks the same name so we only had to compile our code once, but we put them in different directories to keep the input and output files for each chunk separate.) This approach has the advantage that each chunk of comets finishes in far less time than running a multi-threaded simulation with 1000 comets split among many CPUs. We had to use this approach because, in addition to the desktop machines in the physics department, we also ran simulations on the GradEA and MPC Beowulf clusters on campus. These clusters use a queue scheme, with each job allowed a limited amount of time to run. In some cases we could run 100 comets, 10, or, in some cases, only 1 comet in the time allotted. We wrote more Linux scripts to combine the data for each of these smaller simulations to the form it would have had if we had instead run all 1000 comets together. We could have figured out the random seed that would be necessary to generate each grouping of comets instead of reading this data from a file, but we consider the latter approach safer.

Parallelizing our code using the second method is not as ideal as the first because it forces each simulation to repeat a number of calculations for the disk, bulge, and halos that are the same for each chunk of comets, but it did allow us to use many more CPUs than we otherwise could have used, which considerably sped up the execution of slow simulations. The fast simulations (like those without halos) were run in serial since they ran quickly enough that they would have been slowed down considerably by the overhead involved in splitting a program into threads.

7 Results

We first present the initial distributions of the initial conditions for the comets and halos; in these sections we also assess the validity of some of the assumptions we used to construct our model. Then we present the results of our simulations and, finally, we analyze how Diemand's dark matter halos affect the Oort cloud.

7.1 Initial conditions for the comets

In Fig. 3 we have plotted a histogram of the distribution of the radii from the sun for the 1000 comets we generated. We note that they are all confined to radii between 0.1 pc and 0.5 pc and that the radii are uniformly distributed as we required in §4.3.

In Fig. (5) we checked the azimuthal symmetry of the cloud by making a histogram of the azimuthal coordinate, ϕ; as we expected, the distribution is uniform.

The distribution of polar angles in Fig. 6 follows Eqn. (5) fairly well, with more values of theta near the equator (where $\theta = \pi/2$) where there is more surface area and volume than at the poles (where $\theta = 0$ or π).

When we break the position vector into cartesian components in Figs. 7, 8, and 9 we notice some interesting behavior. There are far more comets with cartesian components near the origin then at the edges of the Oort cloud. The reason for this is that a sphere has more of its area and volume where x, y, and z are small than where x, y, and z are at their extremes. We are comforted that the distributions for x, y, and z appear roughly identical since it means the distribution of comets is spherically symmetric.

In Fig. 4 we have plotted a histogram of the distribution of the magnitudes of the velocity vectors. From Eqn. (4) we note that this distribution depends on the distribution of radii as plotted in Fig. 3. The mean speed is 133.44 m/s, the standard deviation is 45.674 m/s, the median is 123.19 m/s, the minimum is 57.068 m/s, and the maximum is 264.27 m/s.

When we break the velocity vector into components in Figs. 10, 11, and 12 we first notice that the distributions appear roughly identical and we also notice that small velocities are far more common than large ones. The non-uniformity of these distributions point to why randomly picking initial speeds will cause a large number of comets to escape: they are simply moving too fast.

To further check our results we also created plots with many thousands of comets and found that the distributions smoothed out considerably (as one would expect) into beautiful curves and had the shapes we required in §4.3.

7.2 Initial conditions for the halos

We first plot in Fig. 13 a histogram of the distributions of impact parameters of the 50122 halos generated when the halos' mass is 10 $M_\odot$ (we choose this mass so we don't generate more halos than Microsoft Excel can handle). The median impact parameter with respect to the sun is roughly 2.5981 pc, the mean is 2.3572 pc with a standard deviation of 0.6694 pc, the minimum is 0.029123 pc, and the maximum is 3.0000 pc. For comparison, the radius of the solar system is roughly 40 AU $\approx$ 0.00001933 pc from the size of Pluto's orbit. The number of halos that enter the outer barrier of the Oort cloud, 0.5 pc, and the number that enter the solar system is, from Fig. 13, very small; thus the approximation that the halos can be treated as point objects is a good one in view of the gravitational shell theorem.

In Figs. 15 and 16, we plot the initial azimuthal and polar angles, respectively, of the halos. These distributions have the same shape as those for the comets since we randomly distributed the objects across the surface of a sphere in both cases. However, the distribution of X, Y, and Z coordinates differ from the comets' distribution because the halos were all generated at the same radius while the comets were generated at randomly chosen radii. The distribution of cartesian coordinates is uniform since we covered the area of the 3 pc sphere as uniformly as possible.

We also plotted the velocity distribution for the halos in Fig. 14. The minimum speed is 8.4573 km/s, the maximum is 256.524 km/s, the median is 147.388 km/s, the mean is 144.059 km/s, and the standard deviation is 41.770 km/s. The maximum speed is consistent with Eqn. (18). In Figs. 20, 21, and 22 we break the velocity vectors into components. We find it comforting that the distributions for V_x, V_y, V_z are virtually identical.

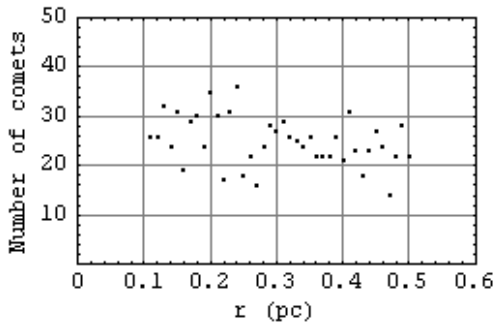

Figure 3: Initial distribution of the comets' radii from the sun

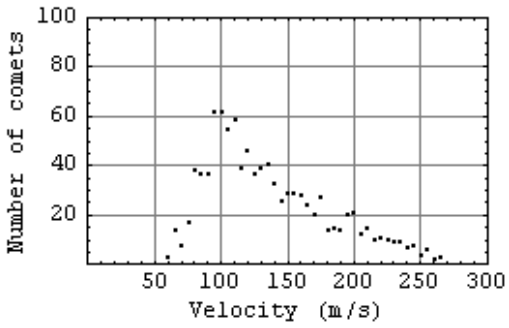

Figure 4: Initial distribution of the comets' velocities

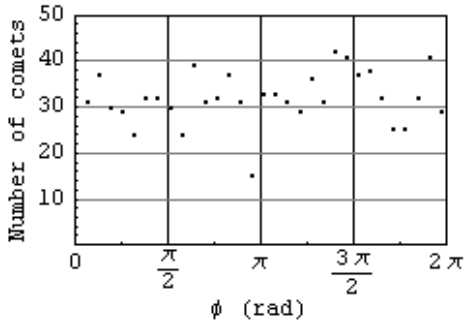

Figure 5: Initial distribution of the comets' azimuthal angles

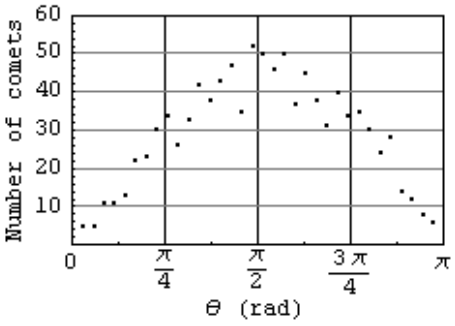

Figure 6: Initial distribution of the comets' polar angles

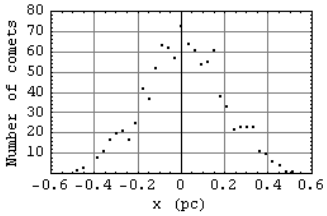

Figure 7: Initial distribution of the comets' x coordinates

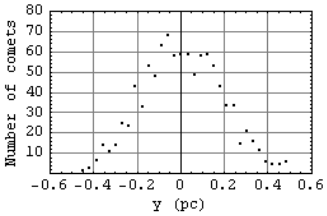

Figure 8: Initial distribution of the comets' y coordinates

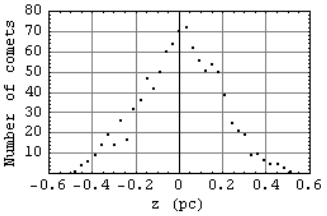

Figure 9: Initial distribution of the comets' z coordinates

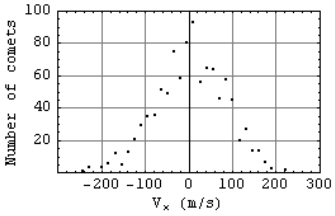

Figure 10: Initial distribution of the comets' V_x coordinates

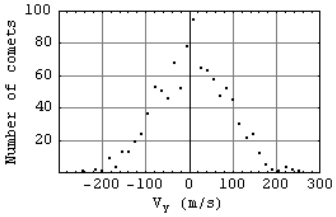

Figure 11: Initial distribution of the comets' V_y coordinates

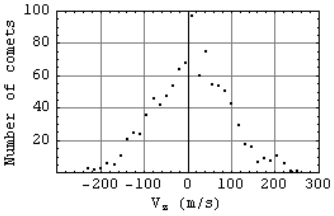

Figure 12: Initial distribution of the comets' V_z coordinates

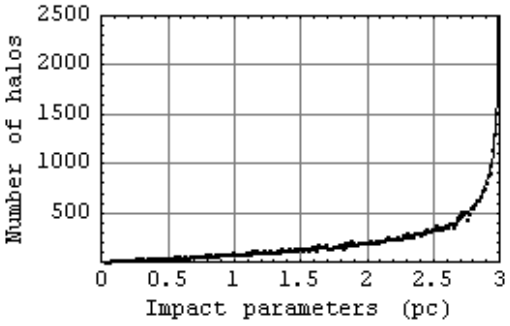

Figure 13: Histogram of the halos' impact parameters, plotted with Eqn. (25).

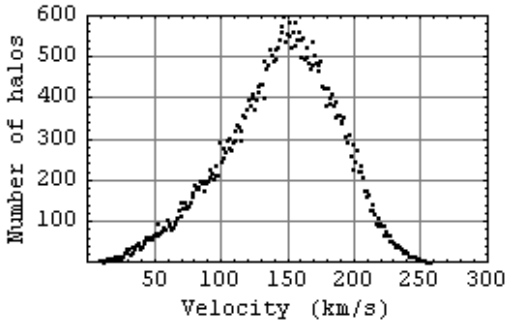

Figure 14: Initial distribution of the halos' velocities

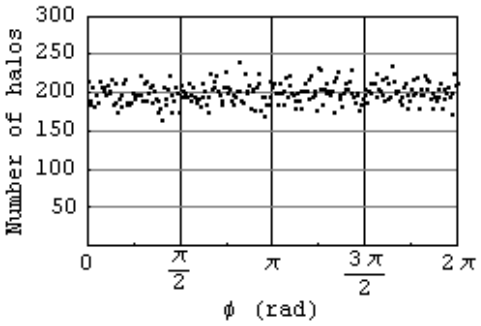

Figure 15: Initial distribution of the halos' azimuthal angles

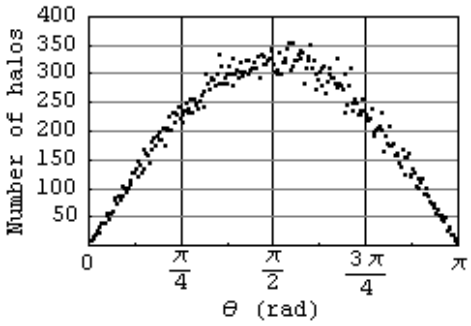

Figure 16: Initial distribution of the halos' polar angles

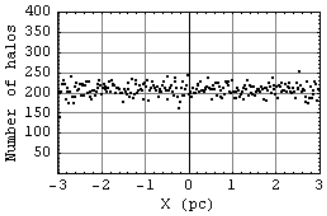

Figure 17: Initial distribution of the halos' X coordinates

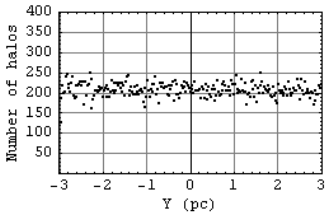

Figure 18: Initial distribution of the halos' Y coordinates

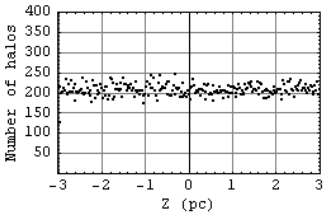

Figure 19: Initial distribution of the halos' Z coordinates

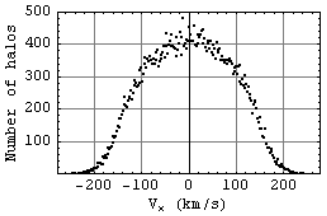

Figure 20: Initial distribution of the halos' V_X coordinates

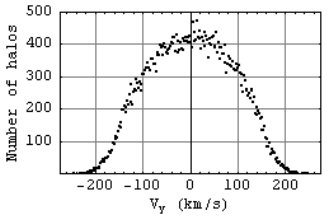

Figure 21: Initial distribution of the halos' V_Y coordinates

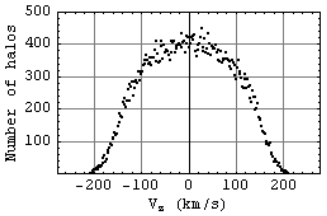

Figure 22: Initial distribution of the halos' V_Z coordinates

7.3 Halo time interval, count, density

We first produce a table containing the values we used for the time interval between halos from Eqn. (31), with the values for M_{encl} and t coming from Eqns. (28) and (29), respectively. Since the smallest unit of time in our simulations is a year, we round all values down, which will increase the number of halos and so increase their disruptive influence on the Oort cloud.

Table 3: Time interval between halos as a function of halo mass

Halo mass ($M_\odot$)	Time interval (years)
10	20164
1	2016
0.1	201
0.01	20
0.001	2

The speed of our simulations is strongly correlated with the number of halos since each halo has to be advanced in time and interacts with each comet. The simulation with 10 $M_\odot$ halos ran in about a day on a single CPU, but when we parallelized the simulation with 0.001 $M_\odot$ halos and ran it on at least 50 CPUs, it took about a month and a half to complete. Because we have access to a limited number of CPUs and do not have the time to wait, we were not able to run simulations with lighter halos than those listed in Table 3.

Next we plot in Fig. 23 the number of halos as a function of time and halo mass, and then we list some related statistics in the table that follows.

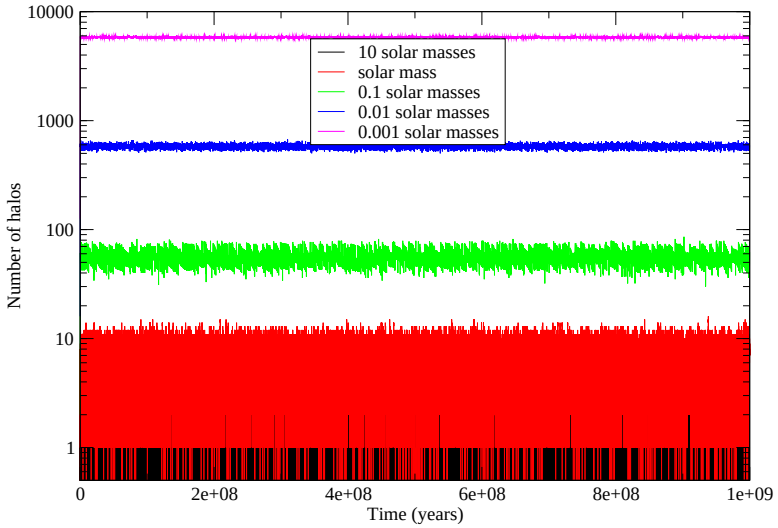

Table 4: Statistics of the distribution of the number of halos as a function of halo mass

Halo mass ($M_\odot$)	median	mean	standard deviation
10	0	0.58242	0.6536
1	6	5.7699	2.0199
0.1	58	57.906	6.3919
0.01	582	581.90	20.372
0.001	5816	5815.7	66.360

The number of halos active at any given time can be turned into a mass density by multiplying the number of halos by their mass and dividing by the volume available to them (see Eqn. (27)). We plot a histogram of the density of dark matter as a function of the halos' mass in Fig. 24 and list some related statistics in the table that follows.

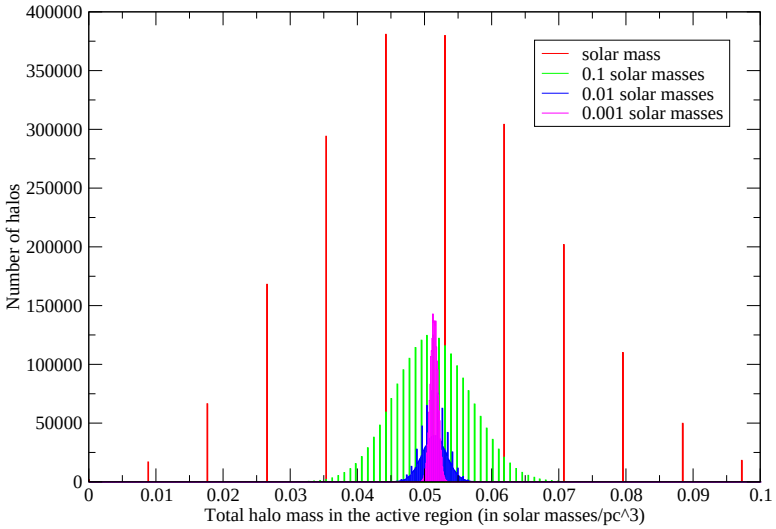

Figure 24: Histogram of the density of dark matter in the 3 pc sphere containing the halos as a function of the halos' mass. The target density is 0.05 $M_\odot/pc^3$. The histogram with 10 $M_\odot$ halos has been left out since 1 halo creates a mass density out of the range of the plot (roughly 0.9 $M_\odot/pc^3$); there are usually either 0 halos or 1 halo.

Table 5: Statistics of the distribution of the total mass of active halos as a function of halo mass

Halo mass ($M_\odot$)	median ($M_\odot/pc^3$)	mean ($M_\odot/pc^3$)	standard deviation ($M_\odot/pc^3$)
10	0	0.051496	0.057787
1	0.053050	0.051016	0.017859
0.1	0.051282	0.051198	0.0056516
0.01	0.051459	0.051450	0.0018012
0.001	0.051423	0.051421	0.035390

At this point we are equipped to justify the numbers we used in §4.5.5. The target dark matter density is 0.05 $M_\odot/pc^3$ from Eqn. (26) and the values in Fig. 24 and Table 5 match this value well. We are a little on the high side because we rounded the time interval between halos down and used the maximum velocity a halo could have. This magnifies the effects of the halos since they are created more often, but this does not affect the conclusions we will make in the sections that follow. (We ran simulations with much smaller and much larger values of ρ_{DM} to confirm this). The differences between ρ_{DM} and our values are well within the error bars for ρ_{DM}.

7.4 Binned comet data

7.4.1 Sun, disk, bulge

In Fig. (25) we have plotted the population of the Oort cloud (defined as the number of comets within 1 pc of the sun) for several cases without halos. Several features of this plot deserve further comments.

The first comets are not stripped out for almost 70 Myr when the disk is active and about 340 Myr when the bulge is active without the disk. In the first case the time delay between the onset of the perturbation at $t = 0$ and the loss of the first comet is likely due to the time it takes for a comet to travel from inside 0.5 pc to the boundary of the 1 pc net we have cast over the Oort cloud.

The bulge appears to be a few times less effective at removing comets than the disk is, as we anticipated in §3.1.

The disk periodically causes the population to drop rapidly in a short amount of time. The period between these drops is roughly $2 \times T_{1/2}$ from Eqn. (11), corresponding to a full oscillation of the solar system above and below the disk. The periodic rapid drops in population result from the sign change in $f(t)$ in Eqn. (12) when the solar system passes through the disk. Modeling the disk using a step function is inappropriate when the solar system is near the disk since the disk does not seem two-dimensional like it does when the solar system is far from it. A more accurate model of the disk may produce significantly different behavior during the times when the solar system is near the disk.

The shape of the curve with both the sun, disk, and bulge is basically the same as the curve for the disk alone, except that the overall downward trend has been enhanced by the bulge.

We recall from §3 that roughly 40% to 60% of the comets in the Oort cloud are likely to have been destroyed or ejected into space over the life of the solar system. We can see from Fig. (25) that if our simulations were run for 4.5 Gyr (the approximate age of the solar system) the population of the cloud would be roughly consistent with these numbers. (We ran simulations without halos for this amount of time to confirm this). We also note that curves seem to flatten out as time advances, indicating that the comets with the least stable orbits are ejected while comets with more stable orbits are retained by the sun.

7.4.2 Sun, disk, bulge, halos

In Fig. (26) we have plotted the population of the Oort cloud for cases with halos. We note that the overall shape is given by the disk (as we can see by examining Fig. (25)) but the downward trend is more pronounced for heavier halos. The curves look almost the same for halos with masses in the range 0.001 $M_\odot$ to 0.1 $M_\odot$, with the curves crossing each other several times. This indicates that simulations with halo masses over two orders of magnitude are indistinguishable within the errors associated with the force approximations and physical constants we have used.

In Fig. (27) we have combined the two previous plots to compare the effects of all the perturbations to the Oort cloud. The results from the simulation with the disk and bulge but no halos are approximately the same as the results with simulations including halos with masses between 0.001 $M_\odot$ and 0.1 $M_\odot$. This indicates that such light halos do not perturb the Oort cloud much more than the Galactic perturbations and are likely quite small compared to the perturbations we neglected in §3.1. We can conclude on the basis of this plot that the dynamics of the Oort cloud cannot be used to rule out the existence of halos with masses below $M_\odot$.

The Oort cloud is depleted very rapidly for halo masses above $M_\odot$. The rate of depletion far exceeds the rates when no halos are present and far exceeds the estimate that 40% to 60% of the original cloud's population still exists. This plot allows us to conclude that Diemand's halos must be lighter than $M_\odot$. We note that the curves seem to flatten out as time advances, both with halos and without halos. This seems to indicate that comets with the least stable orbits are stripped out while comets with more stable orbits are retained by the sun. It may be possible that comets with certain

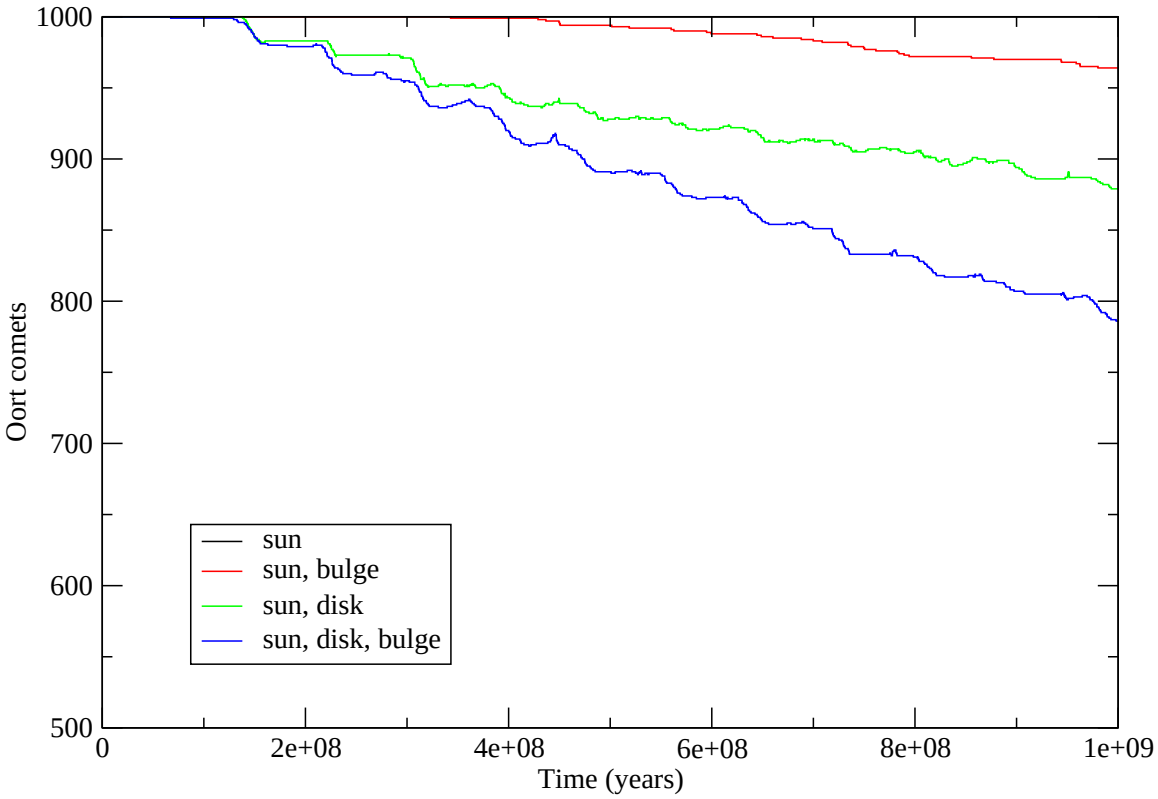

Figure 25: Population of the Oort cloud for simulations with the sun, disk, and bulge. The population for the simulation with only the sun remained constant at 1000; no comets were lost because we created stable orbits. This graph has been zoomed compared to Fig. 27 to show more detail. The periodic drops in population occur when the solar system passes through the disk. The population drops more rapidly when the bulge is added to the disk but the essential shape of the curve comes from the disk.

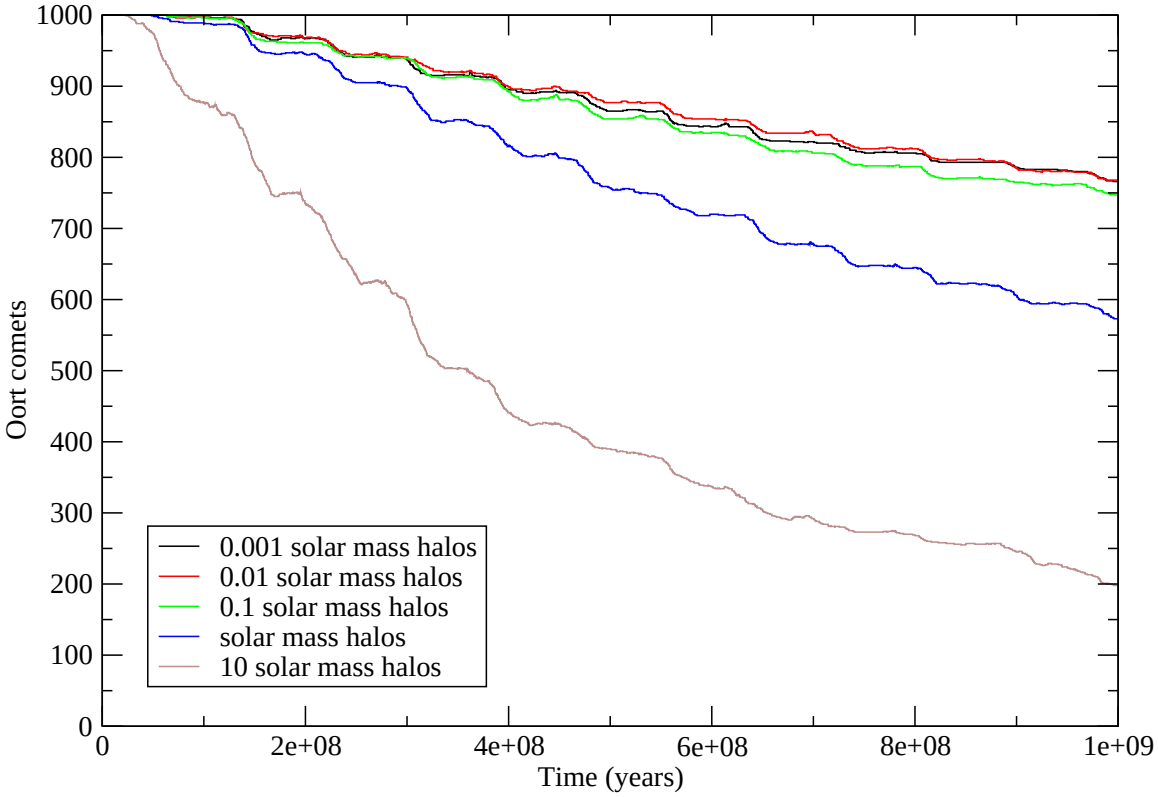

Figure 26: Population of the Oort cloud when the sun, disk, bulge, and halos of different masses are present. The sun, disk, and bulge were present for each simulation but we left them out of the legend so it doesn't cover part of one of the curves.

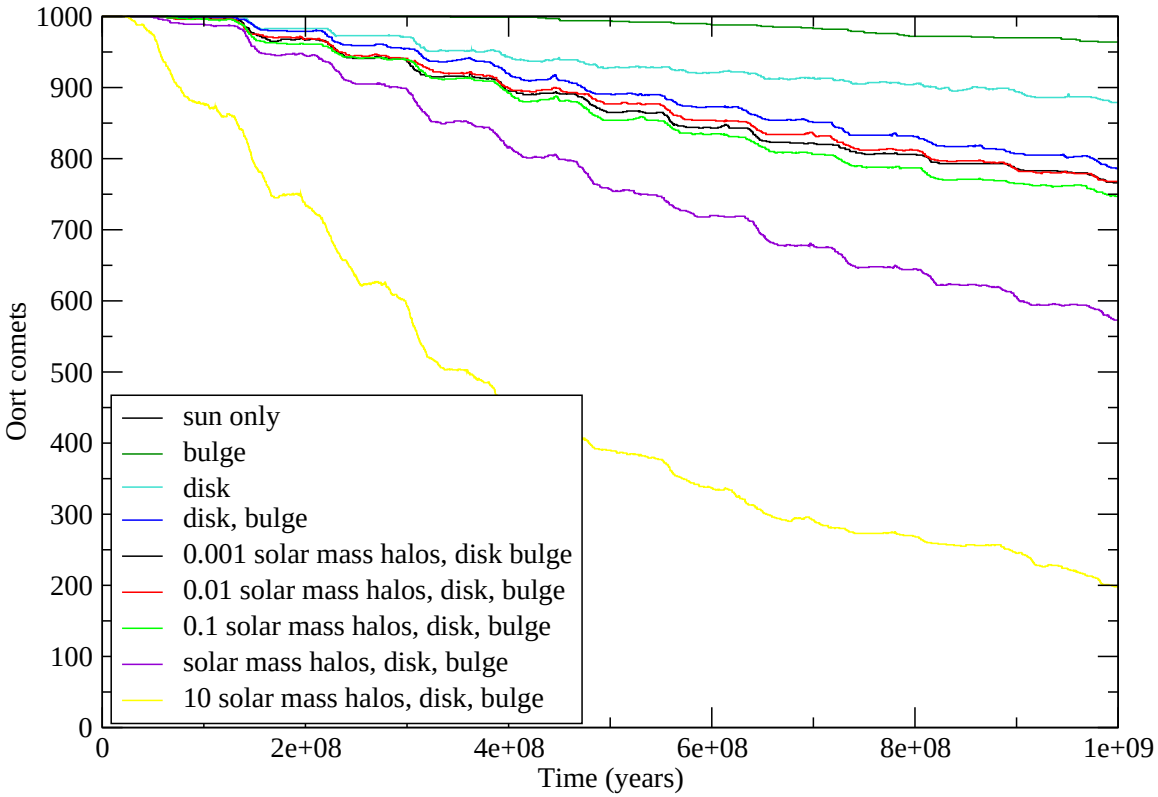

Figure 27: Plot of the population of the Oort cloud with all sources of perturbation. The sun is present for each simulation but we left it out of the legend to make it more compact. The population curves appear to flatten as time advances, indicating that the perturbations have removed comets with the least stable orbits. Comets with more stable orbits are retained by the sun.

7.5 Comets' impact parameters

Fernandez [4] blended the work of several authors and observations to estimate the number of new comets that cross Earth's orbit. His figure is 3 new comets every 4 years, with an error of roughly $\pm 30\%$. He also cites an earlier calculation by Weissman in 1990, who estimated that roughly 10 comets cross Earth's orbit every year. The order of magnitude difference between these two numbers is indicative of the large errors involved in producing such estimates and is evidence of discrepancies among authors. It is important that these are "new" comets since many comets pass by several times before being destroyed or ejected into space, while other comets (e.g., Halley's comet) fall into long period orbits and are observed passing by the Earth many times. Our comets are all "new" comets since we only count their point of closest approach to the sun once.

In §3 we quoted 6×10^{12} comets as an estimate of the total population of the Oort cloud. For each of our 1 Gyr simulations of 1000 comets we therefore expect roughly

$$\begin{gathered} \tfrac{3}{4}\tfrac{\text{comets}}{\text{year}}(1 \times 10^9 \text{ years})\tfrac{1000 \text{ comets}}{6\times 10^{12} \text{ comets}} \approx 0.125 \text{ comets} \\ \text{to} \\ 10\tfrac{\text{comets}}{\text{year}}(1 \times 10^9 \text{ years})\tfrac{1000 \text{ comets}}{6\times 10^{12} \text{ comets}} \approx 1.67 \text{ comets} \end{gathered} \tag{41}$$

cross to cross inside Earth's orbit.

In Table 6 we list the number of comets that can within 40 AU, 100 AU, and 1000 AU of the sun (the table is cumulative in the sense that any comets within 100 AU are within 1000 AU). We recall that Pluto is about 40 AU from the sun and Earth is 1 AU from the sun. Several features in this table are of note.

Table 6: Comets' impact parameters

Simulation type	40 AU	100 AU	1000 AU
Sun only	0	0	0
Disk	0	2	10
Bulge	15	50	168
Disk, Bulge	9	31	105
0.001 $M_\odot$ halos, Disk, Bulge	5	16	61
0.01 $M_\odot$ halos, Disk, Bulge	5	15	63
0.1 $M_\odot$ halos, Disk, Bulge	4	15	63
$M_\odot$ halos, Disk, Bulge	8	22	58
10 $M_\odot$ halos, Disk, Bulge	9	28	90

Many comets enter the solar system but the minimum impact parameter is roughly 10 AU. The fact that no comets came within 1 AU of the sun is consistent with Eqn. (41) since the expected number is so small. Inside the solar system one would need to include the effects of Jupiter and Saturn to accurately model the trajectories of the comets to determine whether any of our comets will cross inside Earth's orbit. When a comet nears the sun it is accelerated considerably so that it could have a velocity that would carry it across the solar system in one 50 year time step. It is therefore possible that a comet did enter within 40 AU (or even 1 AU) of the sun but was missed because the comet moved in and out of 40 AU between adjacent time steps. Hence, we have tabulated the number of comets inside 100 AU and 1000 AU to cast a larger net over the impact parameters.

We constructed orbits that were perfectly stable and were contained between 0.1 pc and 0.5 pc, which is why no comets entered the solar system when just the sun was present.

In §3.1 we pointed out that the disk could not drop the Galacto-centric $\mathcal{Z}$-component of the angular momentum of most comets below a certain value except for rare configurations; this is why the disk alone cannot bring many comets into the solar system.

The bulge alone brought the most comets into the solar system but adding the disk to it tempered the bulge's influence. When we compare this with Fig. 25 we notice that the bulge is capable of bringing more comets into the solar system, while the disk tends to strip more comets out of the cloud. When the two are simulated together, the number of comets entering the solar system drops while the number of ejected comets increases.

Adding halos seems to increase the "gravitational shielding" against the influence of the bulge. More massive halos decreased the influx of comets slightly while lighter halos decreased the flux considerably. There were several orders of magnitude more light halos than heavy halos, giving the comets more sources of attraction. The halos were also *far* closer to the cloud than either the disk or the bulge. We therefore conclude that halos do not increase the influx of comets into the solar system.

7.6 Conclusions

Based on the results presented in §§7.4 & 7.5 we conclude that the approximations that went into writing Eqn. (34) are at least plausible and do not conflict with astronomical observations.

We also conclude that the halos with masses below 0.1 $M_\odot$ do not significantly perturb the Oort cloud so we cannot use the dynamics of the Oort cloud to determine whether or not they can exist. Halos with masses above $M_\odot$ are heavy enough that they strip out too many comets to fit with current observations and theoretical estimates for the population of the Oort cloud; such halos can be ruled out on this basis. Somewhere between 0.1 $M_\odot$ and $M_\odot$ there is a cross-over region where the halos become too light to significantly affect the Oort cloud. It would be a simple matter to determine more carefully what this mass is by running simulations with several masses in this region and by comparing the results with those in Fig. (27).

Another possible course of future research would be to relax the constraint that the halos should all have the same mass in each simulation. It would be simple to randomly pick masses in a range such that Eqn. (26) is maintained. This would allow one to produce a more accurate model of the effects of halos on the Oort cloud, with heavy halos occurring rarely and light halos occurring often. It would also be advantageous to quantify the effects of changing Eqn. (26); if more evidence appears that dark matter should be clumped into halos, this course of research could help to provide a more accurate estimate of Eqn. (26).

To more accurately determine the number of comets that cross Earth's orbit, one would need to include the effects of Jupiter and Saturn; however, on the basis of the data presented in Table 6, the influx of comets is not likely to be a good parameter to put constraints on the mass of Diemand's halos.

In building our model we have assumed the Oort cloud presently exists and all the orbits we generated were initially stable. While this does allow us to study the maximum effects halos could have on the Oort cloud, it does not trace the history of the Oort cloud. An interesting project for someone would be to simulate the formation of the Oort cloud in an environment where massive halos pass by frequently. It may be possible that even light halos disturb the Oort cloud enough

that it would not be able to form. If this were the result of such a study, it would be possible to rule out light halos from consideration.

References

[1] Bullock, J., Kolatt, T., Sigard, Y., Sommerville, R., Kravtsov, A., Klypin, A., Primack, J., Dekel, A. *Profiles of dark haloes: evolution, scatter, and environment.* MNRAS 321, 559-575, 2001

[2] Chakrabarti, S. K. "Properties of the Oort cloud and the origin of comets" *Mon. Not. R. Astr. Soc. (1992)* **259***, 37-46*

[3] Diemand, J., Moore, B., Stadel, J. "Earth-mass dark-matter haloes as the first structures in the early Universe" *http://www.arXiv.org/astro-ph/0501589* January, 2005.

[4] Fernández, Julio Angel. Comets: Nature, Dynamics, Origin, and the Cosmological Relevance. The Netherlands: Springer, 2005.

[5] Goldstein, Herbert. *Classical Mechanics* London: Addison-Wesley Publishing Company, Inc., 1959

[6] Landau, Rubin H., Páez, Manuel J. Computational Physics: Problem Solving with Computers. New York: John Wiley & Sons, Inc., 1997

[7] Masi, Marco. *Dynamical effects of the radial galactic tide on an Oort cloud of comets for stars with different masses and varying distances from the galactic center. http://www.arXiv.org/astro-ph/0403599* March, 2004.

[8] Wackerly, Dennis D., Mendenhall, William III, Scheaffer, Richard L. *Mathematical Statistics with Applications, Fifth ed.* San Francisco: Duxbury Press: An imprint of Wadsworth Publishing Company, An International Thomson Publishing Company., 1996

[9] Weissman, P. R. "The Oort cloud" *Completing the Inventory of the Solar System. ASP Conference Series, Vol. 107, 1996* T.W. Rettig and J.M. Hahn, eds.

[10] Weisstein, Eric W. *"Sphere Point Picking."* from MathWorld – A Wolfram Web Resource. *http://mathworld.wolfram.com/SpherePointPicking.html*

A C++ code

Our computer code appears below. The spacing for some of the lines looks odd because we had to take out extra spaces in some lines, and other lines had to be split in two in order to fit the code within the margins on each page.

```
#include <math.h>
#include <iostream.h>
#include <fstream.h>
#include <iomanip.h>

// physical constants:
const double EarthMass = 5.9723e24; //units:  kg
const double JupiterMass = 1.8987e27; // units: kg
const double SunMass = 1.98844e30;  //units:  kg
const double GalaxyMass = 1.3e11 * SunMass; // units:  kg
const double AU = 149597870660.0; //AU = Astronomical Unit
const double PC = 3.0856775807e16; // units: m / pc
const double G = 6.6742e-11 * pow(AU, -3.0); // grav const in AU^3 / (kg s^2)
const double Pi = 3.141592654;
const double year = 60.0 * 60.0 * 24.0 * 365.0; // one year in seconds
const double SunBulgeDistance = 8500.0 * PC / AU; // 8.5kpc converted to AU
const double RhoDisk = 0.1 * SunMass * pow(AU / PC, 3); // kg / AU^3
const double SunDiskAmplitude = 70.0 * PC / AU; // AU
const double bulgeOmega = 2.0 * Pi / 225000000; // 2pi / 225 Myr

// simulation parameters (all distances are converted to AU):
const double haloMass = SunMass;
const double RhoDM = 0.05 * SunMass;
const double innerRadius = 0.1 * PC / AU; // Oort cloud inner radius
const double outerRadius = 0.5 * PC / AU; // Oort cloud outer radius
const int numComets = 1000;
const double maxDM1DVelo = 150000.0 / AU; // AU/s
const double initialHaloRadius = 3.0 * PC / AU; // starting radius for halos
const double timeStep = 50.0 * year;
const double simTime = 1000000000.0; // total simulation time in years

// data analysis parameters
const double innerBinRadius = 0.0;
const double outerBinRadius = 1.0 * PC / AU; // 1 pc
const double binSize = 0.05 * PC / AU; // (outer - inner) must be divisible by binSize
const int RAMbins = 200; // number of bins stored in RAM
const double binTime = 20000.0; // binning time interval, in years
const double haloCheckInterval = 500.0; // in years
const double trackInterval = 1000000; // how often output::cometTracker() is called, in years

// used repeatedly by comets' force functions:
const double GM = G * SunMass;
const double GGM = G * GalaxyMass;
const double GDM = haloMass * G;
const double timeStepOver2 = timeStep / 2.0;

double setHaloInterval(){ // sets and returns the halo interval in units of years
  double VolDMHaloSphere = (4.0 / 3.0) * Pi * pow(initialHaloRadius * AU / PC, 3);
  double containedDMMass = RhoDM * VolDMHaloSphere; // total DM mass in the sphere (in kg)
  double medianImpactParameter = 2.59 * PC / AU; // from data
  double transitTime = 2.0*sqrt(initialHaloRadius * initialHaloRadius - medianImpactParameter
                                * medianImpactParameter) / (maxDM1DVelo*sqrt(3)*year);
```

```
  double halointerval = (transitTime * haloMass / containedDMMass);
  return floor(halointerval); // round the interval down to an integer number of years
}

inline double randDouble(){ // returns a random double between 0 and 1
  return (double)rand() / (double)RAND_MAX;
}

inline int sign(){ // random sign generator returns either + or -
  if( (float)rand() / RAND_MAX <= 0.5) {return +1;}
  else {return -1;}
}

class DMHalo{
public:
  // public data:
  double x[3], v[3], rSquared; // x[0] = x, x[1] = y, x[2] = z, v[0] = Vx, v[1] = Vy, v[2] = Vz
  // modified by deltaPosition() and used by comet::force() functions:
  double mod2X[3], mod2rSquared, mod3X[3], mod3rSquared;
  // use these when using DMLL.initialData(), otherwise use the ones in the constructor:
  // double impactParameter, itheta, iphi, vtheta, vphi, veloR, veloPhi, veloTheta;
  // double creationtime; static double* time;

  inline DMHalo(){ // constructor initializes position, velocity
    // creationtime = *time; // use this only when you want to use DMLL.initialData()
    // use these when you're not using DMLL.initialData():
    double itheta, iphi, veloR, veloPhi, veloTheta, vtheta, vphi;

    // initializing the position (random theta & phi, but r = outerDMRadius)
    rSquared = initialHaloRadius * initialHaloRadius;
    // randomly distributed spherical coordinates
    iphi = 2 * Pi * randDouble();
    itheta = acos(2 * randDouble() - 1);
    // converting the position to cartesian coordinates
    x[0] = initialHaloRadius * sin(itheta) * cos(iphi);
    x[1] = initialHaloRadius * sin(itheta) * sin(iphi);
    x[2] = initialHaloRadius * cos(itheta);

    // local cartesian system (in spherical coordinates)
    veloR = - randDouble() * maxDM1DVelo;
    veloTheta = sign() * randDouble() * maxDM1DVelo;
    veloPhi = sign() * randDouble() * maxDM1DVelo;
    // angles to orient the spherical unit vectors:
    vphi = 2 * Pi * randDouble();
    vtheta = acos(2 * randDouble() - 1);

    // converting to cartesian coordinates:
    v[0] = veloR * sin(vtheta) * cos(vphi) + veloTheta * cos(vtheta) * cos(vphi)
                                           - veloPhi * sin(vphi); // Vx
    v[1] = veloR * sin(vtheta) * sin(vphi) + veloTheta * cos(vtheta) * sin(vphi)
                                           + veloPhi * cos(vphi); // Vy
    v[2] = veloR * cos(vtheta) - veloTheta * sin(vtheta); // Vz

    // impact parameter of the halo, use only when using DMLL.initialData() since it's slow:
    /* impactParameter = sqrt((pow((x[1]*v[2]-x[2]*v[1]),2) + pow((x[2]*v[0]-x[0]*v[2]),2)
             + pow((x[0]*v[1]-x[1]*v[0]), 2)) / (v[0]*v[0]+v[1]*v[1]+v[2]*v[2])); */
  }
```

```
  inline void advance(){ // advances the position coordinates by a single timeStep.
    // new coordinates were calculated by deltaPosition() and were stored in mod3x[]:
    x[0] = mod3X[0]; x[1] = mod3X[1]; x[2] = mod3X[2];
    rSquared = x[0] * x[0] + x[1] * x[1] + x[2] * x[2];
  }

  inline void deltaPositions(){ // advances the coordinates by timeStepOver2 and timeStep
    static int d; // index is reused each time the function is called
    for (d = 0; d < 3; d++) {
      mod2X[d] = x[d] + v[d] * timeStepOver2;
      mod3X[d] = x[d] + v[d] * timeStep;
    }
    // modified squared radii for timeStepOver2 and timeStep
    mod2rSquared = mod2X[0] * mod2X[0] + mod2X[1] * mod2X[1] + mod2X[2] * mod2X[2];
    mod3rSquared = mod3X[0] * mod3X[0] + mod3X[1] * mod3X[1] + mod3X[2] * mod3X[2];
  }

  DMHalo* prev; // pointers used by DMLL, the linked list of DM Halos
  DMHalo* next;
};
//double* DMHalo::time; // use this when you want to use DMLL.initialData()

class DMLinkedList{ // Dark Matter Linked List
private:
  DMHalo* head; // head and tail pointers for the head and tail of the linked list
  DMHalo* tail;

public:
  int numHalos; // keeps track of how many halos there are

  DMLinkedList(){ // constructor
    numHalos = 0;
    head = NULL; tail = head; // initially there are no halos
  }

  DMHalo* getHead() {return head;} // returns the position of the head of the linked list

  inline void add(){
    static DMHalo* oldTail;
    if (numHalos == 0){ // if the list is empty
      head = new DMHalo;
      tail = head;
      head->prev = NULL; head->next = NULL; // there's nothing before or after the head/tail
    }
    else{ // if the list isn't empty
      oldTail = tail;
      tail = new DMHalo; // new node goes at the end of the list
      oldTail->next = tail;
      tail->prev = oldTail;
      tail->next = NULL; // there's nothing after the tail
    }
    ++numHalos; // adds one halo to the counter
  }

  inline void remove(DMHalo* node){ // removes a DMHalo
    if ((node == head) && (node == tail)){
```

```
    head = NULL; tail = NULL; // the only halo has been removed
  }
  else if ((node == head) && (node != tail)){
    node->next->prev = NULL;
    head = node->next;
    }
  else if ((node == tail) && (node != head)){
    node->prev->next = NULL;
    tail = node->prev;
  }
  else if ((node != head) && (node != tail)){
    node->next->prev = node->prev;
    node->prev->next = node->next;
  }
  --numHalos; // subtracts one halo from the counter
  delete node;
}

inline void advance(){
  static DMHalo* advanceCurrent;
  advanceCurrent = head; // start at the head of the linked list
  while(advanceCurrent != NULL){ // quits at the tail of the list
    advanceCurrent->advance(); // advance that halo by a timeStep
    advanceCurrent = advanceCurrent->next; // go to the next halo
  }
}

inline void deltaPositions(){ // advances the positions by timeStepOver2 and timeStep
  static DMHalo* deltaCurrent;
  deltaCurrent = head;
  while (deltaCurrent != NULL){
    deltaCurrent->deltaPositions();
    deltaCurrent = deltaCurrent->next;
  }
}
/*
inline void initialData(){ // outputs the initial data for the halos
  ofstream outfile; outfile.open("test_initial_DMhalo_data"); outfile << setprecision (10);
  double velo; short i;
  DMHalo* halo = head;
  outfile << "time \t impactParameter \t velo \t x (pc) \t y \t z \t Vx (m/s) \t Vy \t Vz
             \t itheta \t iphi \t vtheta \t vphi \t veloR \t veloTheta \t veloPhi" << endl;
  while(halo != NULL){
    velo = sqrtf(halo->v[0]*halo->v[0] + halo->v[1]*halo->v[1] + halo->v[2]*halo->v[2]);
    outfile<< halo->creationtime <<"\t"<< halo->impactParameter*AU/PC <<"\t"<< velo*AU <<"\t";
    for(i = 0; i < 3; i++) {outfile << halo->x[i]*AU/PC <<"\t";} // positions in pc
    for(i = 0; i < 3; i++) {outfile << halo->v[i]*AU <<"\t";} // velocities in m/s
    outfile << halo->itheta <<"\t"<< halo->iphi <<"\t"<< halo->vtheta <<"\t"<< halo->vphi<<"\t"
            << halo->veloR <<"\t"<< halo->veloTheta <<"\t"<< halo->veloPhi << endl;
    halo = halo->next;
  }
  outfile.close();
}
*/

inline void radiusCheck(){ // removes halos that are too far from the sun
  static DMHalo* radiusCurrent; static DMHalo* radiusTemp;
```

```
    static const double outerDMRadiusSquared = initialHaloRadius * initialHaloRadius;

    if(numHalos != 0) { // for the case when there's at least 1 halo
      radiusCurrent = head;
      while(radiusCurrent != NULL) // quits at the tail
{ // if a halo is too far from the sun, remove it
  if(radiusCurrent->rSquared > outerDMRadiusSquared) {
    //store the next halo temporarily, remove current halo, point to previous halo
    radiusTemp = radiusCurrent->next; // store next halo
    remove(radiusCurrent);
    radiusCurrent = radiusTemp;
  }
  // if the halo's radius is acceptable, advance to the next halo
  else {radiusCurrent = radiusCurrent->next;}
}
    }
  }
};

class comet{
 private:
  static DMLinkedList* DMLL; // pointer to the DMLinkedList object
  // these carry data between RungeKutta() and the force functions:
  double modX[3];
  double forceReturn[3];
  // repeatedly used variables, mostly in the force functions:
  static DMHalo* chead; // stores the head of the DMLinkedList
  static double bulge1X[3], bulge2X[3], bulge3X[3]; // bulge's position at t,t+h/2,t+h
  static double bulgeSun1[3], bulgeSun2[3], bulgeSun3[3]; // vector connecting bulge and sun
  static double diskForce; // force due to the disk
  double DMforce[3], denom, cometRadiusSquared, CRSpow, bulgeDenom, bulgeComet[3], num;
  DMHalo* fcurrent; // pointer to a halo
  // used in the Runge-Kutta function:
  short coord; // for-loop index
  double k1[3], k2[3], k3[3]; // Runge-Kutta trial functions

  inline void force1(){ // force functions for RK() for time = t
    num = 1.0 / sqrt(rSquared);
    CRSpow = - GM / (num * num * num); // denominator of the sun's force

    // rezeros the force caused by the dark matter halos in the last iteration:
    DMforce[0] = 0.0; DMforce[1] = 0.0; DMforce[2] = 0.0;
    // computing the force due to the halos:
    fcurrent = chead; // start at the head of the DM halo linked list
    while(fcurrent != NULL) {// quits at the tail of the linked list or when there are no halos
      // computes and sums the forces between each halo and the current comet
      num = sqrt(rSquared+fcurrent->rSquared -
                 2.0*(x[0]*fcurrent->x[0]+x[1]*fcurrent->x[1]+x[2]*fcurrent->x[2]));
      denom = 1 / (num * num * num);
      DMforce[0] += (x[0] - fcurrent->x[0]) * denom;
      DMforce[1] += (x[1] - fcurrent->x[1]) * denom;
      DMforce[2] += (x[2] - fcurrent->x[2]) * denom;
      fcurrent = fcurrent->next;
    }
    // vector joining the bulge and the comet:
    bulgeComet[0] = bulge1X[0] - x[0];
    bulgeComet[1] = bulge1X[1] - x[1];
```

```
  bulgeComet[2] = bulge1X[2] - x[2];
  // denominator of the bulge's force:
  num = sqrt(bulgeComet[0] * bulgeComet[0] + bulgeComet[1] * bulgeComet[1]
             + bulgeComet[2] * bulgeComet[2]);
  bulgeDenom = GGM / (num * num * num);

  // returns to RK() the sun's force + DM Halo force + Galactic disk force + Galactic bulge force
  forceReturn[0] = x[0] * CRSpow + DMforce[0] * GDM + bulgeComet[0] * bulgeDenom + bulgeSun1[0];
  forceReturn[1] = x[1] * CRSpow + DMforce[1] * GDM + bulgeComet[1] * bulgeDenom + bulgeSun1[1];
  forceReturn[2] = x[2]*CRSpow+DMforce[2]*GDM
                   + bulgeComet[2]*bulgeDenom+bulgeSun1[2]+diskForce*x[2];
}

inline void force2(){ // force function for RungeKutta() for time = t + timeStepOver2
  // radius squared of the comet using the new coordinates sent by RungeKutta()
  cometRadiusSquared = modX[0] * modX[0] + modX[1] * modX[1] + modX[2] * modX[2];
  num = 1.0 / sqrt(cometRadiusSquared);
  CRSpow = - GM / (num * num * num);

  DMforce[0] = 0.0; DMforce[1] = 0.0; DMforce[2] = 0.0;

  fcurrent = chead;
  while(fcurrent != NULL) {
    num=sqrt(cometRadiusSquared+fcurrent->mod2rSquared - 2.0
     *(modX[0]*fcurrent->mod2X[0]+modX[1]*fcurrent->mod2X[1]+modX[2]*fcurrent->mod2X[2]));
    denom = 1 / (num * num * num);
    // using halos' positions at t = t + timeStepOver2 as set by DMLL.deltaPositions():
    DMforce[0] += (modX[0] - fcurrent->mod2X[0]) * denom;
    DMforce[1] += (modX[1] - fcurrent->mod2X[1]) * denom;
    DMforce[2] += (modX[2] - fcurrent->mod2X[2]) * denom;
    fcurrent = fcurrent->next;
  }
  // use the position of the bulge at t = t + timeStepOver2
  bulgeComet[0] = bulge2X[0] - modX[0];
  bulgeComet[1] = bulge2X[1] - modX[1];
  bulgeComet[2] = bulge2X[2] - modX[2];

  num = sqrt(bulgeComet[0]*bulgeComet[0]+bulgeComet[1]*bulgeComet[1]+bulgeComet[2]*bulgeComet[2]);
  bulgeDenom = GGM / (num * num * num);

  forceReturn[0] = modX[0] * CRSpow + DMforce[0] * GDM + bulgeComet[0] * bulgeDenom + bulgeSun2[0];
  forceReturn[1] = modX[1] * CRSpow + DMforce[1] * GDM + bulgeComet[1] * bulgeDenom + bulgeSun2[1];
  forceReturn[2] = modX[2]*CRSpow+DMforce[2]*GDM
                   + bulgeComet[2]*bulgeDenom+bulgeSun2[2]+diskForce*modX[2];
}

inline void force3(){ // force function for RungeKutta() for time = t + timeStep
  // radius squared of the comet using the new coordinates sent by RungeKutta():
  cometRadiusSquared = modX[0] * modX[0] + modX[1] * modX[1] + modX[2] * modX[2];
  num = 1.0 / sqrt(cometRadiusSquared);
  CRSpow = - GM / (num * num * num);

  DMforce[0] = 0.0; DMforce[1] = 0.0; DMforce[2] = 0.0;

  fcurrent = chead;
  while(fcurrent != NULL) {
    // using the positions of the halos at time = t + timeStep
```

```
      num = sqrt(cometRadiusSquared+fcurrent->mod3rSquared-2.0
                 *(modX[0]*fcurrent->mod3X[0]+modX[1]*fcurrent->mod3X[1]+modX[2]*fcurrent->mod3X[2]));
      denom = 1.0 / (num * num * num);
      DMforce[0] += (modX[0] - fcurrent->mod3X[0]) * denom;
      DMforce[1] += (modX[1] - fcurrent->mod3X[1]) * denom;
      DMforce[2] += (modX[2] - fcurrent->mod3X[2]) * denom;
      fcurrent = fcurrent->next; // advance to the next halo
    }
    // use the position of the bulge at time = t + timeStep:
    bulgeComet[0] = bulge3X[0] - modX[0];
    bulgeComet[1] = bulge3X[1] - modX[1];
    bulgeComet[2] = bulge3X[2] - modX[2];

    num = sqrt(bulgeComet[0]*bulgeComet[0]+bulgeComet[1]*bulgeComet[1]+bulgeComet[2]*bulgeComet[2]);
    bulgeDenom = GGM / (num * num * num);

    forceReturn[0] = modX[0] * CRSpow + DMforce[0] * GDM + bulgeComet[0] * bulgeDenom + bulgeSun3[0];
    forceReturn[1] = modX[1] * CRSpow + DMforce[1] * GDM + bulgeComet[1] * bulgeDenom + bulgeSun3[1];
    forceReturn[2] = modX[2]*CRSpow+DMforce[2]*GDM
                     + bulgeComet[2]*bulgeDenom+bulgeSun3[2]+diskForce*modX[2];
  }

  int chanceOneThird(){ // picks 1, 2, or 3 randomly
    double x = randDouble();
    if (x < (1.0 / 3.0)) return 1;
    else if (x > (1.0 / 3.0) && x < (2.0 / 3.0)) return 2;
    else return 3;
  }

public:
  // public data:
  double x[3], r, rSquared; // position coordinates. x[0] = x, x[1] = y, x[2] = z
  double v[3]; // velocity coordinates. v[0] = Vx, v[1] = Vy, v[2] = Vz
  double impactParameter, impactParameterSquared;

  static double* time; // pointer to the current simulation time

  static void setDMLLPointer(DMLinkedList* DMLLPtr){ // sets the resident DMLL pointer
    DMLL = DMLLPtr;
  }
  inline static void setDMLLHead(){ // sets resident chead pointer to the head of the linked list
    chead = DMLL->getHead();
  }

 comet(){ // constructor initializes the random position and velocity vectors
    // Evan Dowling's method: generate stable 2D orbits then rotate them arbitrarily in 3 dimensions
    double r1, r2, a, velo; // apsidal and semi-major axis distances
    double iphi, itheta, vCorrection;

    r = randDouble() * (outerRadius - innerRadius) + innerRadius; // random radius from the sun
    rSquared = r * r;
    impactParameterSquared = rSquared; //initially the point of closest approach is the starting point

    r1 = randDouble() * (r - innerRadius) + innerRadius; // apsidal distances
    r2 = randDouble() * (outerRadius - r) + r;
    a = (r1 + r2) / 2.0; // semi-major axis distance
```

```
  velo = sqrt( (GM/a)*(2.0*a/r-1) ); // implied velocity that's necessary for a stable orbit

  // randomly distributed spherical position coordinates:
  iphi = 2 * Pi * randDouble();
  itheta = acos(2 * randDouble() - 1);

  // converting the position vector to cartesian coordinates:
  x[0] = r * sin(itheta) * cos(iphi);
  x[1] = r * sin(itheta) * sin(iphi);
  x[2] = r * cos(itheta);

  // pick x, y, or z to pick up the "slack" from making the x[] and v[] vectors orthogonal
  switch(chanceOneThird()){
  case 1: { // x
    v[1] = sign() * randDouble();
    v[2] = sign() * randDouble();
    v[0] = - (x[1] * v[1] + x[2] * v[2]) / x[0];
    break;
  }
  case 2: { // y
    v[0] = sign() * randDouble();
    v[2] = sign() * randDouble();
    v[1] = - (x[0] * v[0] + x[2] * v[2]) / x[1];
    break;
  }
  case 3: { // z
    v[0] = sign() * randDouble();
    v[1] = sign() * randDouble();
    v[2] = - (x[0] * v[0] + x[1] * v[1]) / x[2];
    break;
  }
  }
  // correct the velocity to the right magnitude:
  vCorrection = velo / sqrt(v[0] * v[0] + v[1] * v[1] + v[2] * v[2]);
  v[0] *= vCorrection;
  v[1] *= vCorrection;
  v[2] *= vCorrection;
}

 static inline void setBulgeDiskForce(){
  // constants used repeatedly to calculate the comet-disk & comet-bulge forces:
  static const double FourPiGRhoDisk = 4.0 * Pi * G * RhoDisk;
  static const double SqrtFourPiGRhoDiskInYears = sqrt(FourPiGRhoDisk) * year;
  static const double SunBulgeDistanceSquared = SunBulgeDistance * SunBulgeDistance;
  static const double timeStepInYears = timeStep / year;
  static const double timeStepOver2InYears = timeStepOver2 / year;
  static double bulgeSunDenom;
  // bulge's position at time = t
  bulge1X[0] = SunBulgeDistance * cos(bulgeOmega * (*time));
  bulge1X[1] = - SunBulgeDistance * sin(bulgeOmega * (*time));
  bulge1X[2] = - SunDiskAmplitude * sin(SqrtFourPiGRhoDiskInYears * (*time));
  // force that moves the sun due to the bulge at time = t
  bulgeSunDenom = - GGM * pow(SunBulgeDistanceSquared + bulge1X[2] * bulge1X[2], -1.5);
  bulgeSun1[0] = bulge1X[0] * bulgeSunDenom;
  bulgeSun1[1] = bulge1X[1] * bulgeSunDenom;
  bulgeSun1[2] = bulge1X[2] * bulgeSunDenom;
```

```
  // bulge's position at time = t + timeStepOver2
  bulge2X[0] = SunBulgeDistance * cos(bulgeOmega * ( *time + timeStepOver2InYears));
  bulge2X[1] = - SunBulgeDistance * sin(bulgeOmega * (*time + timeStepOver2InYears));
  bulge2X[2] = -SunDiskAmplitude*sin(SqrtFourPiGRhoDiskInYears*(*time+timeStepOver2InYears));
  // force that moves the sun due to the bulge at time = t + timeStepOver2
  bulgeSunDenom = - GGM * pow(SunBulgeDistanceSquared + bulge2X[2] * bulge2X[2], -1.5);
  bulgeSun2[0] = bulge2X[0] * bulgeSunDenom;
  bulgeSun2[1] = bulge2X[1] * bulgeSunDenom;
  bulgeSun2[2] = bulge2X[2] * bulgeSunDenom;

  // bulge's position at time = t + timeStep
  bulge3X[0] = SunBulgeDistance * cos(bulgeOmega * (*time + timeStepInYears));
  bulge3X[1] = - SunBulgeDistance * sin(bulgeOmega * (*time + timeStepInYears));
  bulge3X[2] = -SunDiskAmplitude*sin(SqrtFourPiGRhoDiskInYears*(*time+timeStepInYears));
  // force that moves the sun due to the bulge at time = t + timeStep
  bulgeSunDenom = - GGM * pow(SunBulgeDistanceSquared + bulge3X[2] * bulge3X[2], -1.5);
  bulgeSun3[0] = bulge3X[0] * bulgeSunDenom;
  bulgeSun3[1] = bulge3X[1] * bulgeSunDenom;
  bulgeSun3[2] = bulge3X[2] * bulgeSunDenom;

  // sets the position of and force due to the disk, based on the z-coordinate of the bulge
  if(bulge1X[2] < 0) {diskForce = - FourPiGRhoDisk;}
    else {diskForce = FourPiGRhoDisk;}
}

inline void RK(){ // executes the Runge-Kutta algorithm for each comet
  static const double onesixth = 1.0 / 6.0;
  // Runge-Kutta step 1:
  force1(); // executes force() at time = 0 with the original coordinates
  for(coord = 0; coord < 3; coord++){ // cycle through x, y, z
    k1[coord] = timeStepOver2 * forceReturn[coord];
    modX[coord] = x[coord] + k1[coord]; // modifies the positions for step 2
  }
  // Runge-Kutta step 2:
  force2(); // executes force() at time = timeStepOver2
  for(coord = 0; coord < 3; coord++){
    k2[coord] = timeStepOver2 * forceReturn[coord];
    modX[coord] = x[coord] + k2[coord]; // sets modX[] for Runge-Kutta step 3
  }
  // Runge-Kutta step 3:
  force2(); // executes force() at time = timeStepOver2
  for(coord = 0; coord < 3; coord++){
    k3[coord] = timeStep * forceReturn[coord];
    modX[coord] = x[coord] + k3[coord]; // sets modX[] for Runge-Kutta step 4
  }
  // Runge-Kutta step 4 & update:
  force3(); // executes force() at time = timeStep
  for(coord = 0; coord < 3; coord++){
    /* updates the velocity with 2 * k1[] & 4 * k2[] to compensate for the extra 1/2 in the
       timeStepOver2 in their assignments. k4[] is not assigned, it just used here:  */
    v[coord] += onesixth * (2.0*k1[coord]+4.0*k2[coord]+2.0*k3[coord]+timeStep*forceReturn[coord]);
    // updates the position coordinates using the new velocity coordinates
    x[coord] += timeStep * v[coord]; // x = x0 + v * t
  }
  // updates r^2 and the impact parameter squared if the comet has moved closer to the sun
  rSquared = x[0] * x[0] + x[1] * x[1] + x[2] * x[2];
  if (rSquared < impactParameterSquared) {impactParameterSquared = rSquared;}
```

```
  }

  // calculate the comet's radius from the sun from the radius squared
  inline void calcCometsRadius(){
    r = sqrtf(rSquared);
  }

  inline void calcImpactParameter(){ // calculate the impact parameter from its square
    impactParameter = sqrtf(impactParameterSquared);
  }
};
// allocates space for the static members of the comet class:
DMHalo* comet::chead;
DMLinkedList* comet::DMLL;
double comet::diskForce;
double* comet::time;
double comet::bulge1X[3], comet::bulge2X[3], comet::bulge3X[3];
double comet::bulgeSun1[3], comet::bulgeSun2[3], comet::bulgeSun3[3];

class bins{ // RAM bins used to store data in RAM before writing it to the disk
private:
  int numBins; // number of data bins (not the class name)
public:
  double binTime; // the time that the bins were filled
  int binnedComets; // number of binned comets
  int* radiiBin; // binned data the bins (class) stores

  // sets all the bins to zero before they're incremented in output::binCometData()
  inline void zero(){
    for(int a = 0; a < numBins; a++) {radiiBin[a] = 0;}
  }

  bins(){
    numBins =  (int)(outerBinRadius / binSize - 1); // number of data bins
    radiiBin = new int[numBins];
    zero();
  }
};

class output{ // handles input/output to/from files
private:
  comet* cometGridPointer; // pointer to the comet array
  int numBins; // number of data bins
  bins binArray[RAMbins]; // number of RAM (class) bins
  ofstream binoutfile; // output file for the bins
  int binIndex; // number of bins currently full

public:
  static double* time; // current time in the simulation

  // constructor initializes variables, output files, and prints some relevant info
  output(comet* cometGridPtr){
    cometGridPointer = cometGridPtr;
    numBins = (int)(outerBinRadius / binSize - 1);
    cout << setprecision (11);
    binIndex = 0;
```

```
  // initialize and empty the output files:
  binoutfile.open("binned_comet_data");
  binoutfile << setprecision (11);
  ofstream cometOut; cometOut.open("comet_track_data"); cometOut << setprecision(11);
  cometOut << numComets << "\t" << numComets * simTime / trackInterval << endl;
  cometOut.close(); // initializes & empties the comet_track_data file
  ofstream impact; impact.open("comet_impact_parameters"); impact.close();

  ofstream readout; readout.open("README"); // file with relevant info for the simulation
  readout << "Contains instructions on how to read each outputted data file." << endl;

  readout << "binned_comet_data: " << endl;
  readout << "Number of comets: " << numComets << endl;
  readout << "Numbins: " << numBins << endl;
  readout << "binSize = " << binSize * AU / PC << " pc or " << binSize << " AU" << endl;
  readout << "binTime = " << binTime << " years" << endl;
  readout << "inner bin radius: " <<innerBinRadius*AU/PC<<" pc or "<<innerBinRadius<<" AU"<<endl;
  readout << "outer bin radius: " <<outerBinRadius*AU/PC<<" pc or "<<outerBinRadius<<" AU"<<endl;
  readout << "Number of years \t # binned comets \t # escaped comets \t comets per bin"<<endl<<endl;

  readout << "initial_comet_grid_data" << endl;
  readout << "[x] = AU, [Vx] = m/s" << endl;
  readout << "comet#   radius   theta   phi   velo   x   y   z   Vx   Vy   Vz" << endl << endl;

  readout << "comet_track_data: " << endl;
  readout << "Comet position and velocity as a function of time: " << endl;
  readout << "x   y   z  for each comet at each time interval in units of AU" << endl << endl;

  readout << "comet_impact_parameters" << endl;
  readout << "number of comets within 40AU (solar system radius), 100AU, 1000AU" << endl;
  readout << "comet impact parameters sorted into bins with the binned_comet_data bins"<<endl;
  readout << "raw comet impact parameters, unsorted, in units of AU" << endl << endl;

  readout << "Simulation constants:" << endl;
  readout << "time step: " << timeStep / year << " years" << endl;
  readout << "halo mass: "<<haloMass<<" kg"<<" or "<<haloMass/SunMass<<" solar masses"<<endl;
  readout << "halo interval: " << setHaloInterval() << " years" << endl;
  readout << "RhoDM: " << RhoDM << " kg or " << RhoDM / SunMass << " solar masses" << endl;
  readout << "DM Halo generation radius: "<<initialHaloRadius*AU/PC<<" pc  or "
          <<initialHaloRadius<<" AU"<<endl;
  readout << "inner Oort cloud radius: "<<innerRadius*AU/PC <<" pc  or "
          <<innerRadius <<" AU"<< endl;
  readout << "outer Oort cloud radius: "<<outerRadius*AU/PC <<" pc  or "
          <<outerRadius<<" AU"<< endl;
  readout << "max halo velocity: " << maxDM1DVelo  * AU << " m/s" << endl << endl;

  readout.close();
}

~output(){
  //  flush any full bins to the hard disk when the program terminates
  int m; // repeatedly used for-loop index
  if(binIndex != 0){
    for(int f = 0; f < binIndex; f++){
     binoutfile <<binArray[f].binTime<<"\t"<<binArray[f].binnedComets
                <<"\t"<<numComets-binArray[f].binnedComets<<"\t";
     for(m = 0; m < numBins; m++) {binoutfile << binArray[f].radiiBin[m] << "\t";}
```

```
     binoutfile << endl;
    }
  }
  binoutfile.close();
}

// write the initial conditions in a form that's appropriate for output::populateGrid() to read:
inline void cometGridParallelData(){
  ofstream outfile; outfile.open("initialCometGridDataForParallel");
  // cycle through the comets, printing the initial position and velocity vectors:
  for(int i = 0; i < numComets; i++){
    outfile << cometGridPointer[i].x[0]<<"\t"<<cometGridPointer[i].x[1]
            << "\t"<<cometGridPointer[i].x[2]<<"\t";
    outfile << cometGridPointer[i].v[0]<<"\t"<<cometGridPointer[i].v[1]<<"\t"
            << cometGridPointer[i].v[2]<<"\t";
    outfile << cometGridPointer[i].rSquared << endl;
  }
  outfile.close();
}

inline void cometGridInitialData(){ // outputs the initial conditions of all the comets
  ofstream outfile; outfile.open("initial_comet_grid_data");
  int j; // repeatedly use for loop index
  for (int i = 0; i < numComets; i++){ // cycles through the comets in the grid
    outfile << i << "\t"; // outputs the comet number
    outfile << cometGridPointer[i].r << "\t"; // outputs the radius
    // spherical theta coordinate, then spherical phi coordinate
    outfile << atan(sqrt(cometGridPointer[i].x[0]*cometGridPointer[i].x[0]+cometGridPointer[i].x[1]
                   *cometGridPointer[i].x[1])/cometGridPointer[i].x[2])<<"\t";
    outfile << atan(cometGridPointer[i].x[1] / cometGridPointer[i].x[0]) << "\t";
    // initial speed of the comet
    double velo = sqrt(cometGridPointer[i].v[0]*cometGridPointer[i].v[0]+cometGridPointer[i].v[1]
                       *cometGridPointer[i].v[1]+cometGridPointer[i].v[2]*cometGridPointer[i].v[2]);
    outfile << velo * AU << "\t"; // outputs the velocity in m/s since it's stored as AU / s
    for (j = 0; j < 3; j++) {outfile <<cometGridPointer[i].x[j]<<"\t";} // position vector (AU)
    for (j = 0; j < 3; j++) {outfile <<cometGridPointer[i].v[j]*AU<<"\t";} // velocity vector (m/s)
    outfile << endl;
  }
  outfile.close();
}

inline void cometTracker(){ // outputs the positions of the comets periodically for later analysis
  ofstream outfile; outfile.open("comet_track_data", ios::app);
  static int i,j;
  for(i = 0; i < numComets; i++){ // cycles through all the comets
    for(j = 0; j < 3; j++) {outfile << cometGridPointer[i].x[j] << "\t";} // outputs the positions
  }
  outfile << endl;
  outfile.close();
}

inline void binCometData(){ // bins the number of comets by radius from the solar system
  static int j,m,f;
  static int binnedComets; // population of the Oort cloud = # comets with radii < 1 pc from the sun
  binnedComets = 0; // rezero this each time the function is called

  for (j = 0; j < numComets; j++) { // cycles through all the comets in the grid
```

```
    if (cometGridPointer[j].r < outerBinRadius) { // the comet must be in a bin to be counted
        // determines which bin the comet falls into and adds one comet to that bin:
      binArray[binIndex].radiiBin[ (int)(cometGridPointer[j].r / binSize) ]++;
      ++binnedComets;
    }
    else {} // if the comet is outside the bins it will be counted as an escaped comet
  }
  binArray[binIndex].binnedComets = binnedComets; // stored data a RAM bin
  binArray[binIndex].binTime = *time; // stores what time the comets were binned

  if(binIndex == RAMbins - 1){ // when all the bins in the binArray are full, output their data
    for(f = 0; f < RAMbins; f++){
      binoutfile <<binArray[f].binTime<<"\t"<<binArray[f].binnedComets<<"\t"
                 <<numComets-binArray[f].binnedComets/*= # escaped comets*/<<"\t";
      for(m = 0; m < numBins; m++) {binoutfile << binArray[f].radiiBin[m] << "\t";}
      binArray[f].zero(); // rezero the RAM bin so it can be reused
      binoutfile << endl;
    }
    binIndex = 0; // rezero the number of full bins after writing the data to the hard disk
  }
  else {++binIndex;} // if there are still empty bins increment the # of full bins counter
}

void cometImpactParameters(){ // outputs the impact parameters from the comets
  int radiiBin[numBins]; // bins for the binning the impact parameters
  // number of comets inside each radius from the sun:
  int solarComets = 0, AU100Comets = 0, AU1000Comets = 0;

  // bin the comets by their impact parameters
  for(int b = 0; b < numBins; b++) {radiiBin[b] = 0;} // zeros the bins
  // the impact parameters are <= original radii, so all of them will fall into one of the bins
  for(int c = 0; c < numComets; c++) {
    radiiBin[ (int)(cometGridPointer[c].impactParameter / binSize) ]++;
    // add # comets less than these radii to the counters:
    if(cometGridPointer[c].impactParameter < 40) {solarComets++;}
    if(cometGridPointer[c].impactParameter < 100) {AU100Comets++;}
    if(cometGridPointer[c].impactParameter < 1000) {AU1000Comets++;}
  }
  // outputs the data to a file
  ofstream outfile; outfile.open("comet_impact_parameters", ios::app);
  outfile << solarComets << "\t" << AU100Comets << "\t" << AU1000Comets << endl;
  for(int e = 0; e < numBins; e++) {outfile << radiiBin[e] << "\t";}
  outfile << endl << endl;
  // outputting the actual comet impact parameters:
  for(int f = 0; f < numComets; f++) {outfile << cometGridPointer[f].impactParameter << endl;}
  outfile << endl; outfile.close();
}

void populateGrid(comet* cometGridPtr){ // reads the initial conditions for the comets from a file
  ifstream infile; infile.open("InitialParallelData");
  for(int i = 0; i < numComets; i++){ // cycle through the comets
    infile >> cometGridPtr[i].x[0]; // position vector
    infile >> cometGridPtr[i].x[1];
    infile >> cometGridPtr[i].x[2];
    infile >> cometGridPtr[i].v[0]; // velocity vector
    infile >> cometGridPtr[i].v[1];
    infile >> cometGridPtr[i].v[2];
```

```
      infile >> cometGridPtr[i].rSquared;
      cometGridPtr[i].impactParameterSquared = cometGridPtr[i].rSquared;
    }
    infile.close();
  }
};
double* output::time; // allocates space for this static member of the output class


int main(){
  srand(100); // seeds the random number generator with the same seed for each run
  comet cometGrid[numComets]; // creates & initializes the comets
  comet* cometGridPointer; cometGridPointer = &cometGrid[0]; // pointer to the array of comets
  DMLinkedList DMLL; // creates an object of type DMLinkedList
  comet::setDMLLPointer(&DMLL); // send the DMLL pointer to the comets
  output out(cometGridPointer); // creates an output class and sends it the pointer to the comets

  // reads in the initial conditions of the comets from a file, if desired:
  // out.populateGrid(cometGridPointer);
  // outputs the initial conditions for the comets to files, if desired:
  // out.cometGridInitialData();
  // out.cometGridParallelData();
  // set the seed in the right spot before creating halos in case the comets were read from a file:
  srand(10100);

  const double haloInterval = setHaloInterval(); // set the time interval between halos
  const double timeStepInYears = timeStep / year;
  int i; // repeatedly used for-loop index
  double time = 0.0, nextBinTime = binTime, nextAddTime = haloInterval, nextAdvance = timeStepInYears,
         nextHaloCheck = haloCheckInterval, nextTrackTime = trackInterval; // all in units of years
  // passes the pointer to the current time to the comet, output, and halo classes
  comet::time = &time; output::time = &time;
  // DMHalo::time = &time; // use only when you want to use DMLL.initialData()

  out.binCometData(); // bin the comets at time = 0
  // begin the simulation:
  for (time = 0.0; time <= simTime; time++){
    if (time == nextAddTime) {
      DMLL.add(); // add a halo
      comet::setDMLLHead(); // give the comets a pointer to the head of the linked list
      nextAddTime += haloInterval;
    }
    if (time == nextAdvance) {
      comet::setBulgeDiskForce();
      DMLL.deltaPositions(); // compute the positions of the halos at time=t+timeStepOver2, t+timeStep
      // the #pragma directives split the following for-loop into threads for multiple CPUs
      //      #pragma omp parallel for schedule(dynamic)
      for(i = 0; i < numComets; i++) {cometGridPointer[i].RK();} // integrate the comets' orbits
      //      #pragma omp barrier // wait for all the threads to finish before continuing

      DMLL.advance(); // advance the positions of the halos from t_n to t_{n+1}
      nextAdvance += timeStepInYears;
    }
    if (time == nextBinTime) {
      // compute the comets' radii before binning them:
      for(i = 0; i < numComets; i++) {cometGridPointer[i].calcCometsRadius();}
      out.binCometData(); // bin the comets by radii
```

```
      nextBinTime += binTime;
    }/*
    if (time == nextTrackTime) { // track the positions of the comets for later analysis
      out.cometTracker();
      nextTrackTime += trackInterval;
      }*/
    if (time == nextHaloCheck) { // remove halos that have strayed too far from the sun
      DMLL.radiusCheck();
      comet::setDMLLHead(); // give the comets a pointer to the head of the linked list
      nextHaloCheck += haloCheckInterval;
    }
  }
  for(i = 0; i < numComets; i++) {cometGridPointer[i].calcImpactParameter();}
  out.cometImpactParameters(); // outputs the impact parameters from the comets
  //DMLL.initialData(); // use to write the initial conditions of the halos to a file
  return 0;
}
```