Numerical Stability Study of a Galactic Stellar Disc

Adrian M. Partl

Abstract

In this bachelors thesis, the effect of a penetrating massive object on the galactic thin disk has been studied using numerical simulations. It was possible to show, that penetrating object are effecting the galactic disk, by heating it up, but it was impossible to make this effect responsible for the evolution of the thin disc to the thick disc.

1 Introduction

For many years, students of astronomy have been taught, that our Milky Way galaxy can be divided into three parts - bulge, disc, and halo. This is not fundamentally wrong, but, as Gilmore & Reid proposed in 1983, a more distinguished view is needed. By analysing the density distribution of stars vertically to the galactic plane, they could identify two distinct features (Gilmore & Reid 1983). Their data showed a sudden change in the density gradient at around 1.5 kpc altitude, dividing the density profile of our disc into two parts. They called these two regions the "old disc" (now being called the "thick disc") and the "thin disc". Since then we had to subdivide the disc into two components and gained a new view of our galaxy as shown in Figure 1.1.

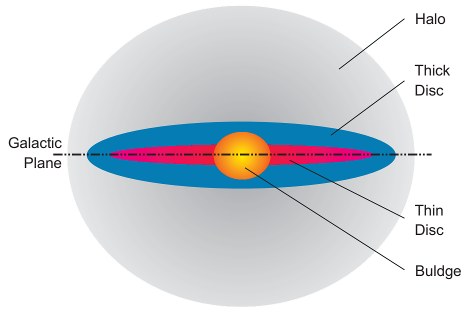

Figure 1.1: Schematic view of the Milky Way Galaxy (Buser 2000)

Another observational indicator can be used to show the existence of features in our galaxy. Stars that were formed at an earlier epoch in the history of a galaxy should not show a high metallicity (for example Fe/H). Younger stars however show higher metallicity since they contain of gas that has already been processed in stars before. A detailed study of this relation has been recently published by (Nordström et al. 2004). It can be seen in their Figure 27, that stars up to the age of about 3 Gyr show a correlation between age and $[Fe/H]$ in such a way that older stars show an abundancy in iron. Stars older than 3 Gyr do not show such a correlation, they are scattered throughout age-metallicity space below the value for solar metallicity of 0 dex. This indicates, that stars younger that 3 Gyr must have had some kind of shared history for a correlation to show. It has been proposed by (Gilmore et al. 1989) that this could be explained with a simple model of constant supernova rates in the ratio 1.5:1.0 for Type I:Type II supernovae.

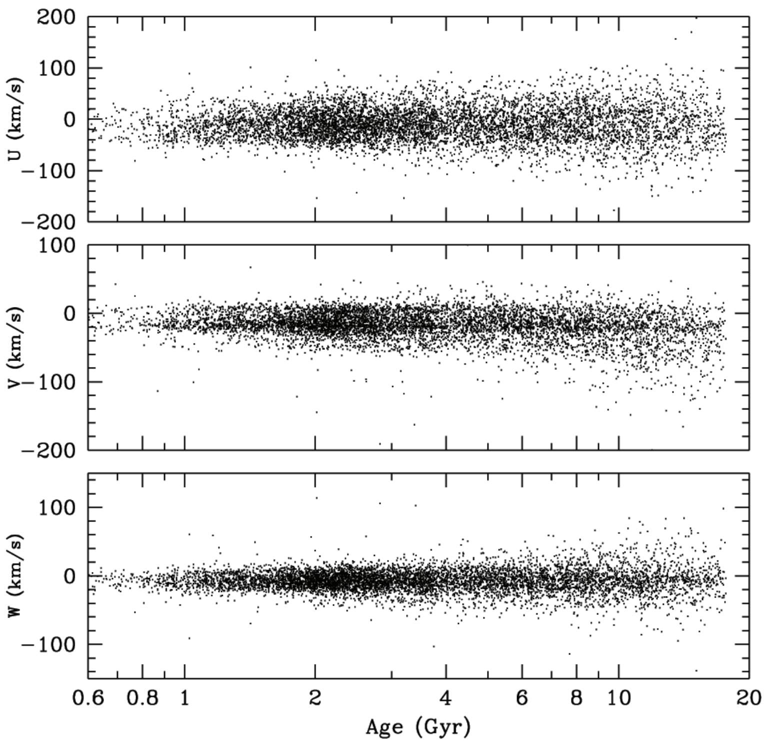

Figure 1.2: U, V and W velocities in relation to age of all sample stars. With increasing age of the stars, the velocity increases. This is visible in the fanning out of the distributions to the right. (Nordström et al. 2004)

Many theories surfaced on how these disc components could have evolved. Several theories have been discussed of which the three most important are: the different discs are leftovers of past quasi-static phases during the collapse of the protostellar cloud to the galactic disc (Gilmore et al. 1989). Another tries to explain the thick disc with gas aggregated from galaxies that merged or just interacted with the Milky Way (Gilmore 2003). Yet another theory proposes a heating mechanism for the thin disc, where parts of the thin disc could have expanded to the thick disc by means of massive objects penetrating the galactic plane (Wielen et al. 1992). These objects could be massive black holes (Wielen et al. 1992) or cold dark matter halos (Font et al. 2001) in orbit around the galactic center. This is the theory that served as an inspiration for this work and served as a challenge to see, whether this effect could be simulated using just a simple N-body algorithm and a ordinary computer.

1.1 Effect of an Irregular Galactic Potential on the Disc

In 1977, R. Wielen was searching for a way to explain the correlation between the velocity dispersion of stars, and their time of formation. The older a star is (in regard to its time of formation), the higher is its velocity. This can be seen in Figure 1.2, taken from the Geneva-Copenhagen survey by Nordström et al. (2004). Freeman & Blend-Hawthorn (2002) provide following values for the vertical velocity dispersion σ_z: for stars younger than 3 Gyr $\sigma_z \sim 10km\,s^{-1}$, stars between 3 and 10 Gyr $\sigma_z \sim 20km\,s^{-1}$, and stars older than 10 Gyr $\sigma_z \sim 40km\,s^{-1}$.

Wielen tried to explain this phenomena by arguing, that the galactic potential should not be considered flat and smooth. Instead he introduced an irregular galactic potential, but refrained from stating the causes for these irregularities. Using this, he derived a theory of diffusion in velocity space and could find diffusion orbits that match observations quite well. The main idea was, that a star moving in an irregular galactic potential gains energy from these irregularities, thus resulting in higher velocities. One has to keep in mind, that at the time Wielen formulated this theory, the thick disc had not been discovered yet as an individual entity.

One way of creating an irregular potential consists of taking some very massive objects and let them pass through the galactic disc: This will heat up the disc and results in expansion of the thin disc. These objects could be massive black holes. In this work, the influence of a massive object on the thickness of the galactic disc is simulated, by using the NBODY0 algorithm developed by Aarseth and described in (Aarseth 2001) and (Aarseth 2003).

The use of an N-body simulation has some advantages over an also appropriate hydrodynamic approximation. By using a hydrodynamic approximation, isotropy needs to be assumed. In reality the particles in the galactic disc show an elongated velocity ellipsoid, showing that isotropy is not approperiate. A direct consequence by just applying a hydrodynamic model would be that a heating up of the stellar "gas" will occur if energy is added to the system. Since in a hydrodynamic world, the penetrating particle needs to be exchanged with some kind of energy source, an increase in heat is the direct consequence. It is thus interesting, if this occurs in an N-body world (as it should) or not. Another advantage of a N-body simulation is, that the effect of a single penetrating particle can be studied. In a hydrodynamic view, there is no existence of one single particle, since hydrodynamics is all about statistics. The penetrating particle can only be expressed by blurring it into some kind of energy source, thus changing the characteristics of the penetrator. Because of all this, a N-body simulation was found to be appropriate. It is convenient to take a closer look at the algorithm used in this simulation.

2 Predictor-Corrector Integration Method

2.1 Basics

Different ways of solving the N-body problem exist. For our considerations we used the predictor-corrector method as has been heavily employed by Aarseth in his works on the N-body problem. We will try to understand the principles of solving N-body problems and will gradually develop the integration scheme.

First it is important to take a closer look at the physics underlying N-body problems. The problem can be simplified such, that only gravitational forces will be interacting with simulated particles. It is thus evident, that Newton's Law of Gravity (2.1) will come to effect in a slightly modified way.

$$\ddot{\vec{r_i}} = -G \sum_{j=1;j\neq i}^{N} \frac{m_j\left(\vec{r_i}-\vec{r_j}\right)}{\left|\vec{r_i}-\vec{r_j}\right|^3} \tag{2.1}$$

This describes the force of all particles j, affecting particle i. G is the gravitational constant. For convenience we will define $G = 1$, resulting in scaled units. In this system, the left side of equation (2.1) can be defined as the force per unit mass $\vec{F_i}$ on particle i. To facilitate reading, particle indices will be omitted and only used when necessary. A softening parameter ϵ is introduced which will prevent particles from coming too close together. This will lead to softened collisions and thus prevents a division by zero. We transform equation (2.1) respectively and bring it into the more convenient form

$$\vec{F_i} = -G \sum_{j=1;j\neq i}^{N} \frac{m_j\left(\vec{r_i}-\vec{r_j}\right)}{\left(r_{ij}^2+\epsilon^2\right)^{3/2}} \tag{2.2}$$

with r_{ij} being the separation of particle i to j.

These are the principles upon which the predictor-corrector scheme has been built. Each single particle has to be characterised with certain attributes. In the most basic case, these are the mass of the particle m, its velocity vector $\vec{v}$ and its position vector $\vec{r}$. Since the mass is assumed to be constant and is defined by the initial conditions, a method of calculating the position and the velocity needs to be developed. This can be done through simply integrating (2.2) which will give us a system of highly coupled differential equations that can only be solved analytically for $N = 2$.

$$\begin{aligned}
\ddot{\vec{r_i}}(t) &= \vec{F_i} \\
\vec{v_i}(t) &= \vec{F_i}\Delta t + \vec{v_i}(t_0) \\
\vec{r_i}(t) &= \frac{1}{2}\vec{F_i}\Delta t^2 + \vec{v_i}(t_0)\Delta t + \vec{r_i}(t_0) \qquad (2.3)
\end{aligned}$$

Δt shall be $\Delta t = t - t_0$, i.e. a suitably small time interval. The aim of the following considerations will be the development of an algorithm to determine the functional development of $\vec{F}$. At many points in these reflections it is being tried to minimise redundancy in calculation. This might make things not clear at first thought. It is very important to eliminate redundancy as much as possible to reduce calculation times, since the code will be iterated many times, multiplying processing time of every redundant operation by millions.

2.2 Force Polynomial

The general idea of determining the changes of $\vec{F}(t)$ on a certain particle is, to take the force at four epochs in the past and present (usually the last three evaluated forces at the epochs t_3, t_2, and t_1, and the current force at t_0) and fit a polynomial through these force points. It is then possible to extrapolate the force to an epoch $t_0 + \Delta t$ in the future, i.e. we *predict* the force applied on this one particle. Figure 2.1 clarifies this method.

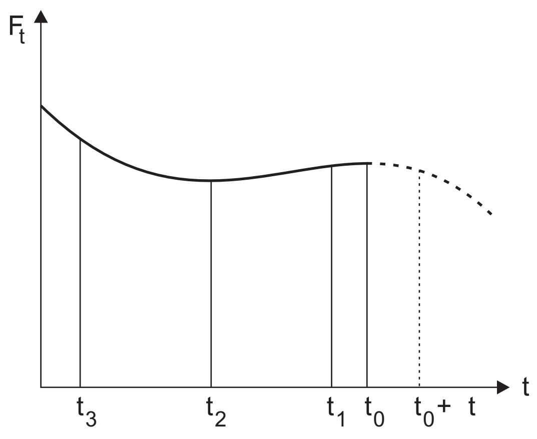

Figure 2.1: Principle of predicting the force applied on one particle using a force polynomial

This approach can only be applied, if the force varies only smoothly and not suddenly. Of course it is not necessary to limit the force polynomial to the fourth order. Higher order polynomials have been used by others, but four orders have been shown to be sufficient for most purposes. Knowing the force $\vec{F}$ at the four epochs in the past t_3, t_2, t_1, and t_0 with t_0 being the most recent, we can formulate the fitting polynomial at time t as

$$\vec{F} = \left\{ \left[\left(\vec{D^4} \left(t - t_3\right) + \vec{D^3} \right) \left(t - t_2\right) + \vec{D^2} \right] \left(t - t_1\right) + \vec{D^1} \right\} \left(t - t_0\right) + \vec{F_0} \tag{2.4}$$

Since the fitting method is based on divided differences, using compact notation, these differences $\vec{D^k}$ are defined as

$$\vec{D^k} \left[t_0, t_k\right] = \frac{\vec{D^{k-1}} \left[t_0, t_{k-1}\right] - \vec{D^{k-1}} \left[t_1, t_k\right]}{t_0 - t_k}, \text{with } (k = 1, 2, 3) \tag{2.5}$$

where $\vec{D^0} \equiv \vec{F}$. The square brackets refer to the corresponding time intervals, for which the difference is being evaluated.

2.3 Further Development of the Method

Until now, we have developed the basic idea of the predictor-corrector integration scheme by using a fitting polynomial and then extrapolating (*predicting*) the force at the time epoch $t_0 + \Delta t$. This idea still needs to be transformed into a more usable form to construct the algorithm. Further, a way to utilise our initial conditions and initialise the algorithm has to be found.

By expanding equation (2.4) into a Taylor series, simple expressions for integration can be gained. For this, we introduce $t'_k = t_0 - t_k$. Equation (2.4) will then look like

$$\vec{F} = \left\{ \left[\left(\vec{D^4} t'_3 + \vec{D^3} \right) t'_2 + \vec{D^2} \right] t'_1 + \vec{D^1} \right\} t'_0 + \vec{F_0}. \tag{2.6}$$

The terms of the Taylor-series can be obtained by differentiating equation (2.6) often enough and adding the pre-factors to each term. For the first term this yields

$$\vec{F^{(1)}} = \left[\left(\vec{D^4} t'_3 + \vec{D^3} \right) t'_2 + \vec{D^2} \right] t'_1 + \vec{D^1} \tag{2.7}$$

which is expanded

$$\vec{F^{(1)}} = \vec{D^4} t'_3 t'_2 t'_1 + \vec{D^3} t'_2 t'_1 + \vec{D^2} t'_1 + \vec{D^1}.$$

Through a second differentiation with respect to t'_3, t'_2, and t'_1 we will obtain

$$\vec{F^{(2)}} = 2! \left[\vec{D^4} \left(t'_1 t'_2 + t'_2 t'_3 + t'_1 t'_3\right) + \vec{D^3} \left(t'_1 + t'_2\right) + \vec{D^2} \right]. \tag{2.8}$$

Doing this again twice will lead to the last two terms of the Taylor series

$$\vec{F^{(3)}} = 3! \left[\vec{D^4} \left(t'_1 + t'_2 + t'_3\right) + \vec{D^3} \right] \tag{2.9}$$

$$\vec{F^{(4)}} = 4! \vec{D^4} \tag{2.10}$$

These equations can be used to find an expression for $\vec{F}$ by putting $t = t_0$. Further, these terms can be transformed into expressions for the divided differences. A problem arising at later time will be, that the value of $\vec{D^4}$ is not known yet. As can be seen from the equations above, the respective values of $\vec{D^4}$ can be added to the corresponding terms at a later time, after having determined a value for $\vec{D^4}$. This is the *correction* part of the predictor-corrector method. The force is predicted with the first three known divided differences. Then this "imprecise" value is used to determine a value for $\vec{D^4}$ (using equation 2.5) and afterwards the force can be evaluated again, this time using the correction term $\vec{D^4}$, resulting in a more precise value.

2.4 The Initialisation Procedure

It is important to correctly initialise the algorithm, otherwise correct results cannot be achieved. Initialisation will make use of the initial conditions that have to be defined for each problem. The initial conditions of one particle are m_j, $\vec{r_j}$, and $\vec{v_j}$. Now the Taylor series derivatives of (2.2) are formed. Again, to facilitate writing let us define $\vec{R} = \vec{r_i} - \vec{r_j}$ and $\vec{V} = \dot{\vec{R}} = \vec{v_i} - \vec{v_j}$. A detailed look at the derivations of the equations below can be found in A. Equation (2.2) thus transforms with $G = 1$into

$$\vec{F_{ij}} = \frac{-m_j \vec{R}}{R^3} \tag{2.11}$$

The first term of the Taylor series is

$$\vec{F_{ij}^{(1)}} = \frac{-m_j \vec{V}}{R^3} - 3a\vec{F_{ij}} \tag{2.12}$$

with $a = \vec{R} \cdot \vec{V}/R^2$. Summation over all particles N will give the total contribution to the Taylor-series for one particle. The next two Taylor-series terms are

$$\vec{F_{ij}^{(2)}} = \frac{-m_j \left(\vec{F_i} - \vec{F_j}\right)}{R^3} - 6a\vec{F_{ij}^{(1)}} - 3b\vec{F_{ij}} \tag{2.13}$$

$$\vec{F_{ij}^{(3)}} = \frac{-m_j \left(\vec{F_i^{(1)}} - \vec{F_j^{(1)}}\right)}{R^3} - 9a\vec{F_{ij}^{(2)}} - 9b\vec{F_{ij}^{(1)}} - 3c\vec{F_{ij}}. \tag{2.14}$$

with $a = \vec{R} \cdot \vec{V}/R^2$ and

$$b = \left(\frac{V}{R}\right)^2 + \frac{\vec{R} \cdot \left(\vec{F_i} - \vec{F_j}\right)}{R^2} + a^2 \tag{2.15}$$

$$c = \frac{3\vec{V} \cdot \left(\vec{F_i} - \vec{F_j}\right)}{R^2} + \frac{\vec{R} \cdot \left(\vec{F_i^{(1)}} - \vec{F_j^{(1)}}\right)}{R^2} + a(3b - 4a^2) \tag{2.16}$$

Again, a summation over all particles will lead to the total contribution. For equations (2.13) and (2.14), the total force contributions $\vec{F}$ and $\vec{F^{(1)}}$ need to be known, before $\vec{F^{(2)}}$and $\vec{F^{(3)}}$ can be initialised. Therefore it is necessary to implement two separate loops (looping through all particles N). The first one initialises $\vec{F}$ and $\vec{F^{(1)}}$, the second one $\vec{F^{(2)}}$and $\vec{F^{(3)}}$.

We have now developed a procedure for boot-strapping the algorithm. Now the initial time steps should be assigned to each particle. t_0 will be set $t_0 = 0$ and constant time steps over the interval used for initialisation will be assumed. For this, a Δt_i needs to be calculated, using considerations discussed later (2.18/2.19). The past epoch time steps can now be calculated using $t_k = -k\Delta t_i$ with $(k = 1, 2, 3)$.

Since starting values for the divided differences are needed, now would be a good time to calculate them. The divided differences are needed for the integration algorithm discussed later and are also used in equation (2.4) for extrapolation. A way of expressing $\vec{D^1}$, $\vec{D^2}$, and $\vec{D^3}$ has to be found. For this, equations (2.7), (2.8), and (2.9) will be inverted, starting with (2.9) and always omitting $\vec{D^4}$. This leads to

$$\vec{D}^1 = \left(\frac{1}{6}\vec{F^{(3)}}t_1^{'} - \frac{1}{2}\vec{F^{(2)}}\right)t_1^{'} + \vec{F^{(1)}}$$
$$\vec{D}^2 = -\frac{1}{6}\vec{F^{(3)}}\left(t_1^{'} + t_2^{'}\right) + \frac{1}{2}\vec{F^{(2)}}$$
$$\vec{D}^3 = \frac{1}{6}\vec{F^{(3)}} \tag{2.17}$$

from which the starting values can be calculated.

The initialisation process is summarised in Algorithm 1, presented below.

Algorithm 1 Initialisation procedure

```
1. Read $m_j$, $\vec{R_j}$, and $\vec{V_j}$ of all particles
2. Loop (looping index $i$) through all particles $N$ to calculate $\vec{F_i}$ and $\vec{F_i^{(1)}}$
    (a) Loop (looping index $j$) through all particles $N$, calculating the force of
        each particle $j$ on particle $i$ using equation (2.11) and (2.12). You will
        get $\vec{F_{ij}}$ and $\vec{F_{ij}^{(1)}}$
    (b) Summation over contributions of all particles $j$ will result in total force
        $\vec{F_i}$ and $\vec{F_i^{(1)}}$
3. Loop (looping index $i$) through all particles $N$ again to calculate $\vec{F_i^{(2)}}$ and
   $\vec{F_i^{(3)}}$
    (a) Loop (looping index $j$) through all particles $N$, calculating terms $\vec{F_{ij}^{(2)}}$ and
        $\vec{F_{ij}^{(3)}}$ using equations (2.13) and (2.14)
    (b) Summation over contributions of all particles $j$ will give total $\vec{F_i^{(2)}}$ and
        $\vec{F_i^{(3)}}$
4. Determine time-steps for current and past epochs using equation (2.18) or
   (2.19) for each particle and calculate the starting value for the divided differ-
   ences using (2.17)
```

2.5 Individual Time Steps

Now that we know how initialisation is achieved, it is good to take a closer look at the integration process. But before doing this, some thought should be given to the use of time steps. To determine the force at some point in the future, the polynomial fitting function (2.4) is used and extrapolated to a time $t_0 + \Delta t$. This Δt is what we call a time step.

There are several ways to integrate using time steps. The obvious method is to globally advance the time constantly by Δt, so that every particle is evaluated at the same time $t_0 + \Delta t$. This method has some major drawbacks.

Let us consider the path of two particles interacting with each other. For simplification one particle is put into a system of rest. The second particle will pass the other particle in considerable small distance, so the interaction will be strong between the two particles. This is illustrated in Figure 2.2, where the starting position of the second particle is on the upper right hand side. The thick line describes the path, resulting from analytical considerations. The dashed line represents the path that the particle would take by just extrapolating the current fitting function using a global time step. The endpoints of both particles are shown after the global time step Δt. One can see, that the particle is not at the position where it should be after this integration step. Extrapolation has been carried out way over the region where the fitting function can be considered similar to the "correct" function, thus causing substantial error. The error is especially big in close encounters. To minimize the error resulting from global time steps, very small global time steps are needed throughout the simulation, resulting in (very) large calculation times.

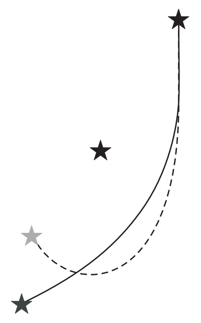

Figure 2.2: Illustrating the individual time step scheme. The thick line is the path a star would take in reality (and with the individual time step scheme) after a time step Δt. The dashed line represents the path that a star would take in simulation with global constant time steps after the same time step Δt.

To minimize errors resulting from approximation by the fitting function, Aarseth introduced in 1963 the individual time step scheme to minimize the effect described above and bringing the fitting function nearer to the "correct" function. The main idea was to have individual time steps for each particle in the simulation and not just one global time step for all. It is then possible to adjust the time steps in the event of a close encounter and thus enhancing precision considerably. A good side-effect of this scheme is, that particles with weak interaction can have bigger time steps, resulting in reduction of calculation time.

Several criteria for calculating the time steps can be used and only two shall be stated here. η is a dimensionless parameter for controlling accuracy of the simulation.

$$\Delta t_i = \left(\frac{\eta \left| \vec{F} \right|}{\left| \vec{F^{(2)}} \right|} \right)^{1/2} \tag{2.18}$$

Equation (2.18) is rather simple and fit for most cases. It produces similar relative errors of the force and leads to similar time steps for strongly interacting particles (Binney & Tremaine 3. edition, 1994). Aarseth developed a more sensitive criterion, including more force derivatives. Through experimentation he was able to find

$$\Delta t_i = \left(\frac{\eta \left(\left| \vec{F} \right| \left| \vec{F^{(2)}} \right| + \left| \vec{F^{(1)}} \right|^2 \right)}{\left| \vec{F^{(1)}} \right| \left| \vec{F^{(3)}} \right| + \left| \vec{F^{(2)}} \right|^2} \right)^{1/2} \tag{2.19}$$

A detailed look at time step criteria and other formulations can be found in (Makino 1991).

2.6 Integration

Integration begins by determining the next particle i to be advanced. The particle j with the smallest value of $t_j + \Delta t_j$ is the next to be integrated. This makes sense since its interaction with the other particles can thus be determined better for particles with a higher value of $t_j + \Delta t_j$. Hence particles are processed chronologically. It is expedient to set global time t to this new value of time $t_j + \Delta t_j$ to reduce redundancy in calculation.

Aarseth introduced in his individual time step scheme two types of coordinates for each particle, namely primary and secondary coordinates r_0 and r_t. The primary coordinates r_0 are being evaluated at time t_0 and represent the actual "precise" coordinates of the particle. The secondary coordinates r_t are based on the primary ones by evaluating the predictor at the time t. The predictor is the function that will be used to predict the position of a particle. For this we will use the Taylor-series expansion of $\vec{r_j}$ only to the order of $\vec{F^{(1)}}$. A higher order could also be used, prolonging calculation accordingly. The Taylor-series expansion of $\vec{r_j}$ in (2.3) to the order of $\vec{F^{(4)}}$ is

$$\vec{r_j} = \frac{1}{720}\vec{F_j^{(4)}}\delta t_j^{'6} + \frac{1}{120}\vec{F_j^{(3)}}\delta t_j^{'5} + \frac{1}{24}\vec{F_j^{(2)}}\delta t_j^{'4} + \frac{1}{6}\vec{F_j^{(1)}}\delta t_j^{'3} + \frac{1}{2}\vec{F_j}\delta t_j^{'2} + \vec{v_0}\delta t_j^{'} + \vec{r_0} \quad (2.20)$$

with $\delta t^{'} = t - t_j$. The expansion to the order of $\vec{F^{(4)}}$ has been given, because we will need it later. Prediction is only carried out to the order of $\vec{F^{(1)}}$. To further reduce redundancy we define $\vec{\tilde{F}} = \frac{1}{2}F$, $\vec{\tilde{F^{(1)}}} = \frac{1}{6}\vec{F^{(1)}}$, $\vec{\tilde{F^{(2)}}} = \frac{1}{2}\vec{F^{(2)}}$, and $\vec{\tilde{F^{(3)}}} = \frac{1}{6}\vec{F^{(3)}}$. The pre-factors in $\vec{\tilde{F^{(2)}}}$ and $\vec{\tilde{F^{(3)}}}$ are obtained trough the pre-factors in (2.8) and (2.9). This will collect some of the pre-factors in (2.20) and let us write

$$\vec{r} = \left(\left(\left(\left(\frac{6}{10}\vec{\tilde{F^{(3)}}}\delta t^{'} + \vec{\tilde{F^{(2)}}}\right)\frac{1}{12}\delta t^{'} + \vec{\tilde{F^{(1)}}}\right)\delta t^{'} + \vec{\tilde{F}}\right)\delta t^{'} + \vec{v_0}\right)\delta t^{'} + \vec{r_0} \quad (2.21)$$

$$\vec{v} = \left(\left(\left(0.75\vec{\tilde{F^{(3)}}}\delta t^{'} + \vec{\tilde{F^{(2)}}}\right)\frac{1}{9}\delta t^{'} + \vec{\tilde{F^{(1)}}}\right)1.5\delta t^{'} + \vec{\tilde{F}}\right)2\delta t^{'} + \vec{v_0} \quad (2.22)$$

Now that all values of $\vec{r_j}$, including the current particle i, have been predicted to the order of $\vec{F^{(1)}}$, particle i has to be improved to the order of $\vec{F^{(3)}}$ by just adding the appropriate missing terms of equation (2.21) to the already calculated expressions. Next the velocity of particle i has to be calculated as well, using (2.22). Now the total force exercised by all the other particles j on particle i has to be calculated through summation of (2.2) over all particles. This is also the time, where any contribution of an external potential, resp. force, needs to be considered.

To be able to form new divided differences, the four times describing the past epoch need to be updated by replacing t_k with t_{k-1} and putting $t_0 = t$. Now the actual integration will take place by forming new divided differences using (2.5), including $\vec{D^4}$. With the new $\vec{D^4}$, a new $\vec{F^{(4)}}$ can be obtained from (2.10). It is now possible to correct the current values of the coordinates and velocity using $\vec{F^{(4)}}$ by collecting all the terms that were omitted until now. Combining all contributions, these correction terms can be found, again using compact notation as above

$$\Delta\vec{r_i} = \vec{F^{(4)}}\left(\left(\left(\frac{2}{3}\delta t^{'} + c\right)0.6\delta t^{'} + b\right)\frac{1}{12}\delta t^{'} + \frac{1}{6}\right)\delta t^{'3}$$

$$\Delta\vec{v_i} = \vec{F^{(4)}}\left(\left(\left(0.2\delta t^{'} + 0.25c\right)\delta t^{'} + \frac{1}{3}b\right)\delta t^{'} + 0.5a\right)\delta t^{'2} \quad (2.23)$$

Algorithm 2 Individual time step cycle

```
1. Determine the next particle: $i = min_j \{t_j + \Delta t_j\}$
2. Set new global time to $t = t_i + \Delta t_i$
3. Predict all coordinates $\vec{r_j}$ of all particles to order $\vec{F^{(1)}}$ (2.21)
4. Form $\vec{F^{(2)}}$ and $\vec{F^{(3)}}$for particle $i$ with (2.8) and $\vec{F^{(3)}} = 6\vec{\vec{F^{(3)}}} = \vec{D^3}$ (for this
   consider 2.9)
5. Improve $\vec{r_i}$ and predict $\vec{v_i}$ to order $\vec{F^{(3)}}$(2.21 and 2.22)
6. Obtain total force $\vec{F_i}$ on particle $i$ by all the other particles
7. Update the times $t_k$ and calculate divided differences using (2.5)
8. Apply the corrector $\vec{D^4}$ to $\vec{r_i}$ and $\vec{v_i}$ (2.23)
9. Calculate new time step $\Delta t_i$ (2.18/2.19)
10. Repeat calculation at step 1
```

The coefficients are $a = t_1^{'} t_2^{'} t_3^{'}$, $b = t_1^{'} t_2^{'} + t_1^{'} t_3^{'} + t_2^{'} t_3^{'}$, and $c = t_1^{'} + t_2^{'} + t_3^{'}$ where the old definition of $t_k^{'}$ still applies. Now the primary coordinates can be updated by setting $r_0 = r_t$ and a new time step needs to be evaluated for particle i.

This is the fourth-order predictor-corrector scheme as developed by Aarseth. A summary of the integration algorithm is provided with Algorithm 2 listed below. To implement the scheme, one requires at least the following 30 variables for each particle: m, $\vec{r_0}$, $\vec{r_t}$, $\vec{v_0}$, $\vec{F}$, $\vec{F^{(1)}}$, $\vec{D^1}$, $\vec{D^2}$, $\vec{D^3}$, Δt, t_0, t_1, t_2, t_3. My implementation uses all the variables used in the scheme (i.e. more than the basic 30 values), making the program easy to read. Saving memory space is not as much an issue today, as it has been when Aarseth developed this scheme.

3 Handling Simulations

3.1 Scaled Units

Processing numbers with a computer comes with some restrictions. It is impossible to store arbitrarily big or small numbers in the computer's memory. Depending on the data type some upper and lower limits exist and restrictions in precision apply. It is therefore necessary to carefully chose the data types when developing a scientific application.

In astronomy, numbers tend to be very big. Hence, some thoughts should be given on how to reduce number size, preventing unwanted buffer overflows. It is obvious, that SI-units are no good choice and other units should be considered. Distances for example should not be given in meters, but parsec is suitable. The same applies to mass, where solar-mass is a quite useful unit. In our simulation we used parsec for distances and solar-masses as the unit of mass.

By putting the gravitational constant $G = 1$, another rescaling of the units has to be done. The gravitational constant contains the units meter, kilogram, and second. If G should be 1, it is not possible to define all these units as one pleases. At least one unit needs to be derived from the other two. Otherwise $G = 1$ is being violated. An example shall be given, how rescaling has been done throughout this paper.

Since the initial conditions for these simulations are the particle mass, position, and velocity, let us use solar-masses for the mass and parsec for the position. It is not possible to choose any unit for the velocity since it is dependent on the time, which will be scaled through the gravitational constant. The value of G in SI-units is $G = 6.672 \cdot 10^{-11} m^3 kg^{-1} s^{-2}$. It is now necessary to transform this value accordingly to the units chosen. This yields $G = 4.5169 \cdot 10^{-30} pc^3 M_{\odot}^{-1} s^{-2}$. Now from this, a conversion factor can be found for one unit of scaled time $1\, unit\, of\, time = 4.705 \cdot 10^{14} s = 1.492 \cdot 10^7 yr$. One unit of time in the simulation is thus $1.492 \cdot 10^7$ years long. This means, the velocity has to be adapted accordingly.

3.2 How to Tell, if the Simulation Runs Correctly?

There is no standardised way of determining, whether a simulation runs correctly or not. Physics supplies laws that need to apply in our simulated world as well. This results in several different ways of determining, whether anything has been done wrong or not. An easy way of monitoring the "correctness" of the simulation is to check, whether energy conservation is being violated or not. This could happen due to low precision in the simulation or fundamental errors in program code.

Energies in the simulation can be calculated using normal physical relations like the equations for the kinetic energy (3.1) or the potential energy (3.2).

$$E_{kin} = \frac{1}{2} \sum_{i=1}^{N} m_i \vec{v_i}^2 \tag{3.1}$$

$$E_{pot} = \sum_{i=1}^{N} \sum_{j>i}^{N} \frac{G m_i m_j}{|\vec{r_i} - \vec{r_j}|} \tag{3.2}$$

In case an external potential is used, the appropriate terms have to be added to the energies. It is very unlikely, that no violation of energy conservation will take place, since every simulation is only an approximation of reality, but the violation should not be significant.

It makes no sense to calculate the energies anew after every integration step. Instead it is convenient to perform the energy check only when outputting data. This is usually done after some time has passed in the simulated reality.

Another good way to test a simulation is to take a scenario that can be solved analytically or where the outcome is already known. For a N-body simulation this could be the orbit of a planet around the sun. But it might not be possible to eliminate all problems with this test. If there is, for example, something wrong with a loop in the code, using only two particles could be too few for coding errors to have significant effects.

3.3 Computation Time and Accuracy Parameter η

Many factors influence computation time. Some of them can be controlled by the user, others are static properties of a computer system as a whole. A big factor in this whole affair is undoubtedly the program code itself. By reducing the redundancy in the calculation, speed can be greatly improved in structures, that have to be processed many times over and over again. This includes the conversion of all fractions to real numbers. Also a good idea is to collect all multiplications and divisions where possible.

Speed can also be gained, by carefully analysing the equations used in the algorithm. Many times, the same expressions shine up in parts of different equations. It is a good idea to calculate these only once, saving them in some temporal variable, and then just use the value of the variable when needed. Of course this can only be done as long as these partial expressions do not change.

Some of these considerations have already been discussed briefly before and resulted in somewhat cryptic but highly optimised equations in Chapter 2. Binney & Tremaine introduced many other small optimisation considerations in their NOBODY0 FORTRAN code, published in the Galactic Dynamics book. This code has been used as a basis for this simulation, and has been adopted accordingly to the C/Objective C programming language. Optimisations resulting in higher speed have been taken over, but measures reducing memory usage were dropped, favoring a code that is easier to read. Memory space is not a big issue anymore today and therefore a factor that can be ignored in this case.

The selection of the accuracy parameter η as introduced in equations (2.18) and (2.19) has also an effect on calculation time. The accuracy parameter can be used to control precision of the simulation. The higher the precision should be, the more calculation time must be applied. Figure 3.1 shows the influence of η on the calculation time. For these measurements, a statically defined system of 10 particles has been used, simulated over the time of one simulation unit. As a measurement of time, iterations have been counted. To compare this with the effect η actually has on the precision, Figure 3.2 has been created, showing the difference in total energy of the system at the beginning and end of the simulation. In a perfect simulation, no difference should arise.

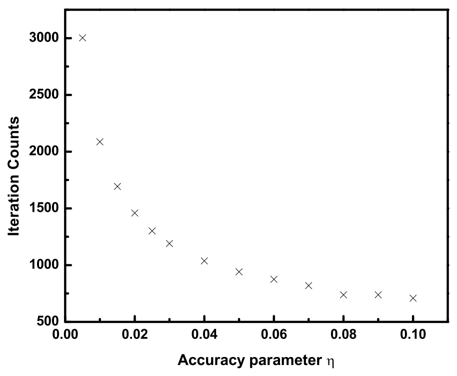

Figure 3.1: The effect of the accuracy parameter η on calculation speed. A statically defined system of 10 particles simulated over one time unit has been used. Iteration counts were used as speed indicator.

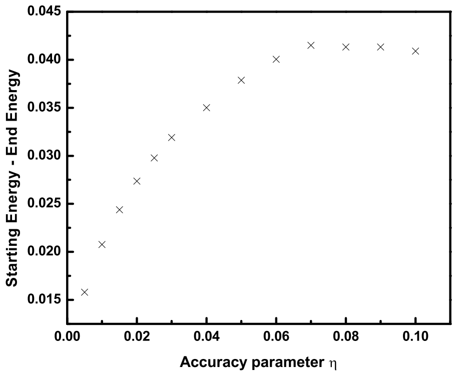

Figure 3.2: The effect of the accuracy parameter η on the accuracy itself. Energy should be conserved, so the difference between the energy at the beginning and at the end gives the calculation error.

The number of simulated particles N play the greatest role in calculation time. Since for every particle, the whole particle system has to be considered, calculation time is roughly proportional to N^2. An additional particle can thus result in significant prolongation of the simulation.

4 Simplifying the Problem and Setting Up the Disc

4.1 Simplifications

Since computation time should be kept at a minimum, further simplification of the problem should be considered. To set up a representative galactic disc, it was tried to simulate only a small portion of the disc. A column of about 10 x 10 pc was considered a useful size, since with about 1000 particles, one particle has around $5M_\odot$ which is not to big. Since only a small portion of the disc was used for the simulation, the velocity gradient that is found in the whole disc should be neglectable in the simulation box. For 10 pc length, this is the case. The point where this column would be in the Milky Way galaxy would be at the position of our sun. This will require some additional simplifications, since we are not simulating the disc as a whole, but only a small part of it.

The influence of the whole disc on this small portion of the disc needs to be handled individually now, thus resulting in the need of an external potential approximating the influence of the whole galactic system. Different forms of potentials can be used. In this work a simple spherical potential has been found appropriate. From radial velocity data of the surroundings of the sun ($r = 8kpc$, $v_c = 220\, km/sec$), a total mass of the galaxy for this radius can be determined, resulting in a mass of $8.9865 \cdot 10^{10} M_\odot$ for our own galaxy. This mass has been used for the external potential, with the radius of 8kpc.

Another way to leverage calculation time and precision is to assume the galactic disc symmetric to the galactic plane. By assuming this, only one side of the disc needs to be considered. To prevent particles from escaping to the other side of the plane, they are mirrored at $z = 0$. This can be done, since it is statistically probable, that with each particle penetrating the galactic plane, another will enter from the other side.

This concept needs to be taken over for the boundaries of the box as well, otherwise all the particles would escape, resulting in drop of density. This would weaken the interaction between the particles. Due to the radial drift of stars and the neglection of a velocity gradient in the small simulation box used, particles that escaped the box have been returned to the system on the other side of the box.

4.2 Distribution of Particles

In order to set up a stable galactic disc, information about several features of a stable disc is required. The particles need to be distributed in a realistic way. Further a correct velocity needs to be determined and assigned to the particles.

Density decreases exponentially in the z-direction of the disc. This can be formulated with a standard exponential function as $\rho(z) = \rho_0 e^{-z/h_z}$, with ρ_0 being the density at $z = 0$ and h_z the scale height. If ρ_0 and h_z are already known, no conversion needs to be done for the density and these values can be used directly in the function. But usually density in the galactic disc is measured as column density Σ, i.e. the total mass located in a column of usually 1 x 1 pc. This needs to be converted into a ρ_0 that can be used in the density function. For this, we take the definition of the column density

$$\Sigma = \int_0^\infty \rho_0 e^{-z/h_z} dz \tag{4.1}$$

and integrate it. The integration term with infinity will disappear, since it is zero. Thus, the only remaining term is $\rho_{col} = \rho_0 h_z$ which can be transformed into an expression for ρ_0.

$$\rho_0 = \frac{\rho_{col}}{h_z} \tag{4.2}$$

Now the particles can be distributed randomly in a sufficiently small interval dz using equations (4.1) and (4.2). The equations are used to determine the amount of mass that should be distributed in the interval dz.

For the simulations a column density of $50\, M_\odot\, pc^{-2}$ has been used. For this, a scale height of about 210 pc (with $\sigma_z \sim 10\, km\, s^{-1}$) has proven to result in a quite stable disc, with only small oscillations due to the external potential. The total mass in the simulation box obtained through equation (4.1) has been distributed evenly over all particles, so that each particle had the same mass. Due to integer divisions in the distribution algorithm a small error in the particle mass of a particle has been introduced. This is why with 1000 particles, the particle mass is not $5M_\odot$ but $4.95M_\odot$ for the system used here.

The other parameter that needs considering while setting up the disc is the velocity of each particle. For a realistic disc of our own galaxy, velocity values should be based on observations. As has been mentioned in 1.1, a good medium value for the thin disc is $\sigma_{tot} \sim 10\,km\,s^{-1}$. It is possible to assign every particle another velocity, scattering it randomly and thus creating an anisotropic velocity ellipsoid. This can be achieved by assigning every particle a random velocity vector, thus pointing in no systematic direction. The length of the vector needs to be corrected to the fixed value of the velocity.

4.3 Simulation Parameters

Now that the disc has been set up, only two parameters need to be specified for the simulation to run. These are the softening parameter ϵ and the accuracy parameter η. There is unfortunately no fixed criteria for choosing values for these two parameters. They need to be determined through experimentation and experience. Care needs only to be taken, that energy is always conserved, otherwise the simulation is more or less useless.

For all simulations, the following values were assumed: $\epsilon = 0.1$ and $\eta = 0.02$.

5 Stable Disc

5.1 Simulation Parameters

For disturbing a galactic disc, it is necessary to set up a stable disc. Stability of a galactic disc in this case is only affected by the scale height and the velocity dispersion. With the simulation parameters mentioned above, a scale height for a more or less stable disc can be determined. Through systematic trial and error, it was possible to determine the value of $h_z = 210\,pc$ for a disc, with only small oscillations around the equilibrium. The layer, where 50% of the total mass are located underneath, only showed oscillations around 30 pc in amplitude. For sure, a better value can be achieved, but these values are sufficiently stable for our considerations.

This setup also seems to be quite stable regarding system energy. The kinetic energy does not show a significant increase, only a very slight one. This is also a indication, that this setup is more or less stable.

5.2 Results

The simulation of a stable disc, as observed in solar surroundings, has been carried out with the parameters discussed throughout this treatise. For convenience, these parameters are summarised below.

System Properties	Simulation Properties
Particle number: ~ 1000 Column density: $50\, M_{\odot}\, pc^{-2}$ Scale height: $210\, pc$ Mass per particle: $\sim 4.95\, M_{\odot}$ Density distribution: exponential Velocity σ_z: $\sim 10\, km\, s^{-1}$ **External Potential:** Spherical Radius: $8000\, pc$ Mass: $8.9865 \cdot 10^{10}\, M_{\odot}$	Softening parameter ϵ: 0.1 Accuracy η: 0.02 Simulation duration: $1.5 \cdot 10^{9}\, yr$

Table 1: Simulation parameters for a stable disc

From the results of the simulation, the kinetic energy has been calculated using (3.1) and from position data, the evolution of the disc in z-direction has been extracted. For this the height of layers with a certain percentage of mass underneath have been evaluated.

In the case of a disturbing particle, a statistical evaluation of the velocity data has been carried out, to determine if any systematic pattern can be seen in the velocity data. For each time in the simulation, a histogram was generated from velocity data. Thus the chronological evolution of the system could be visualised.

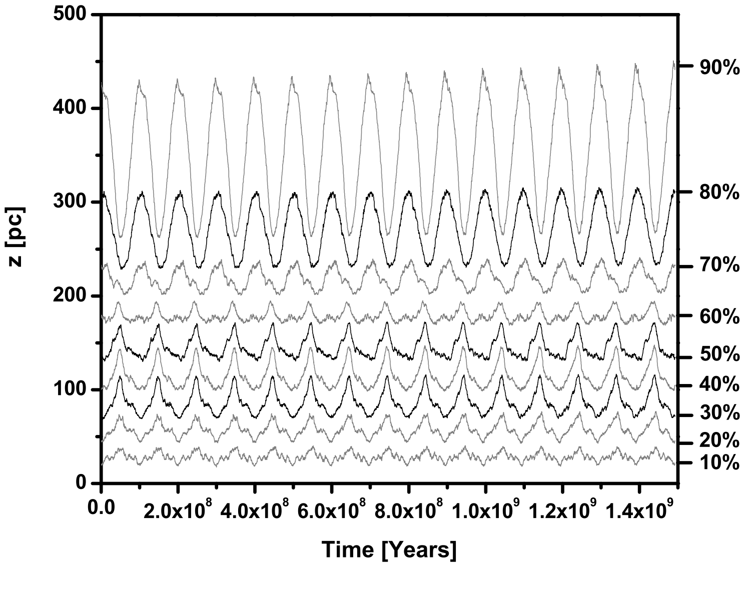

Figure 5.1: Development of a stable disc. Each line in the graph depicts the height, where a certain percentage of mass lies beneath this point. It is evident, that there is almost no expansion over time; thus the disc is stable.

Figure 5.1 illustrates the development of the stable disc over time. Each line corresponds to a height, where a certain percentage of mass lies beneath it. The bold lines are the height where 30%, 50%, and 80% of the mass are situated. The most prominent feature in this graph is definitely the oscillation taking place at all heights.

Since the part of the disc lies near the position where our sun is, the oscillation period should be similar to the analytic value of one particle in a potential field. The analytic value for the oscillation period can be obtained through the following equation which only holds for a single particle in a potential

$$T = \int_{x_1}^{x_2} \frac{dx}{\sqrt{2\left(E - U(x)\right)/m}}$$

where E is the total energy of the system (usually taken at a time, when there is only potential energy, or kinetic energy) and $U(x)$ is the potential energy at point x (only including the external potential, not the potential created by all other particles in the simulation). From this the oscillation period in our potential of $0.88 \cdot 10^8\ yr$ can be found, which corresponds quite well with the simulation.

A slight increase in height can be seen in the 90% line, resulting from hard binary interaction, i.e. particles colliding very closely. During a very close encounter between a binary system and another particle, the initial binary system can get destroyed and one of the partners is ejected with high velocity. A new binary system with the leftover particle from the initial binary and the other particle forms. If this happens more often, the disc can slightly expand, which can be seen at higher altitudes. The disc can still be considered stable, because the lower layers stay constant. This effect is just statistical. These hard binaries will end up in the "Maxwellian tail".

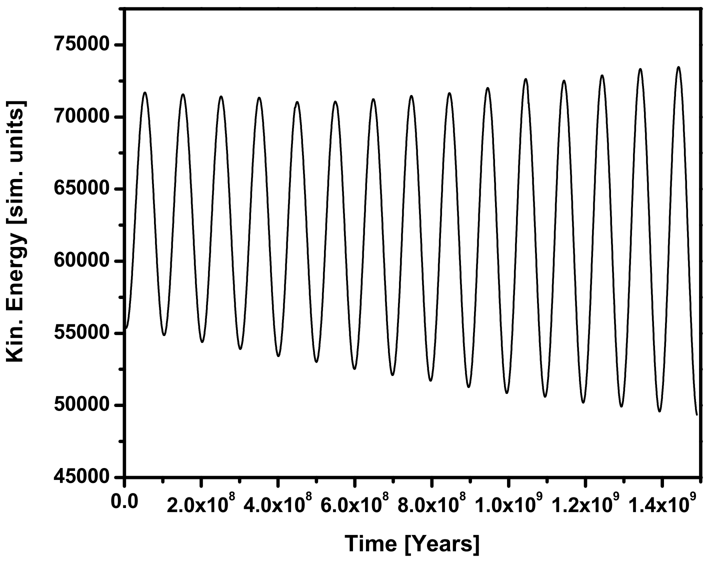

Figure 5.2: Kinetic energy of the system vs. time.

A look at the kinetic energy of the system (Figure 5.2) shows a more or less constant kinetic energy. At first there is a slight drop in energy, corresponding to a damped oscillation. Then it remains constant, with the minima decreasing and the maxima increasing constantly. This increase is the result of hard binary interaction, as mentioned above.

Now that the parameters for a stable disc are known, these can be used, to set up a system that can be disturbed by a massive particle. This will produce the irregular potential that is needed, to study its effects on a galactic disc.

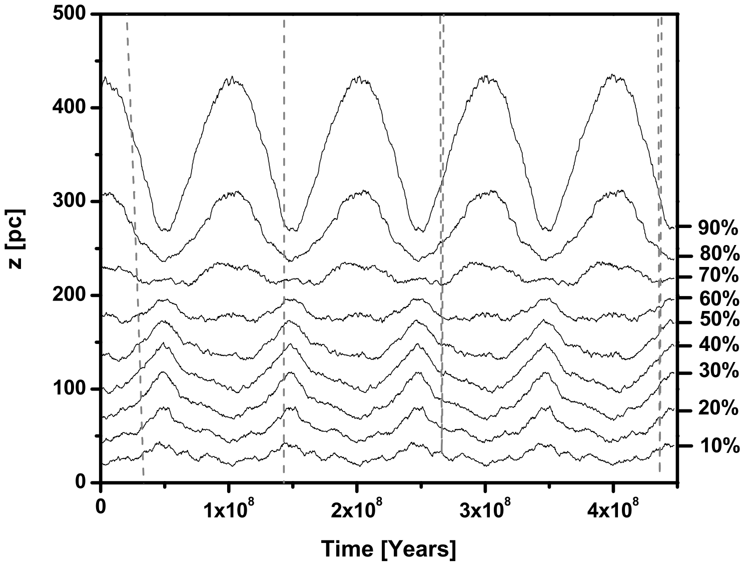

Figure 6.1: Development of a disc, that is being penetrated by a particle with $m = 20000\,M_0$. The external potential was chosen to interact with the disturber. The dashed line represents its path. No significant increase of the layers can be seen.

6 Disturbed Disc

6.1 Problems Arising With Reality

To simulate the effect of a disturbing massive particle on the galactic disc, a system has been set up exactly as described above. For the particle that should penetrate the disc, a mass of $2 \cdot 10^4\,M_\odot$ has been used. Its starting position was at $z = 900\,pc$ above the galactic plane. σ_z of the disturber has been chosen to have the same value as the particles in the disc itself (i.e. $10\,km\,s^{-1}$). The external potential was chosen to interact with the particle.

After a short time of simulation it became clear, that this setup was problematic in regard to the effect of the disturber on the galactic disc. The external particle has been chosen to interact with the external potential, as is the case in reality. This caused the disturber to accelerate and resulted in a very high increase of its velocity, reducing the effect it could have on the disc. The higher the difference of velocity in an encounter is, the smaller is its effect. As is shown in (Binney & Tremaine 3. edition, 1994) using the impulse approximation, this effect of a high speed encounter on the velocity of a particle can be described (for a Plummer model with $a = 0$) with

$$\Delta v_R = -\frac{2GM}{VR}$$

It is thus not surprising, that no major effect on the disc could be seen and the scenario was not simulated completely to the time of $1.5 \cdot 10^9\,yr$. Figure 6.1 shows results of this simulation. The dashed line depicts the path of the disturber. It is clearly visible, that no considerable increase of the disc took place and that the disturber never had time to highly interact with the disc.

Because of these plausible findings the effect of a disturber that does not interact with the galactic potential has been considered.

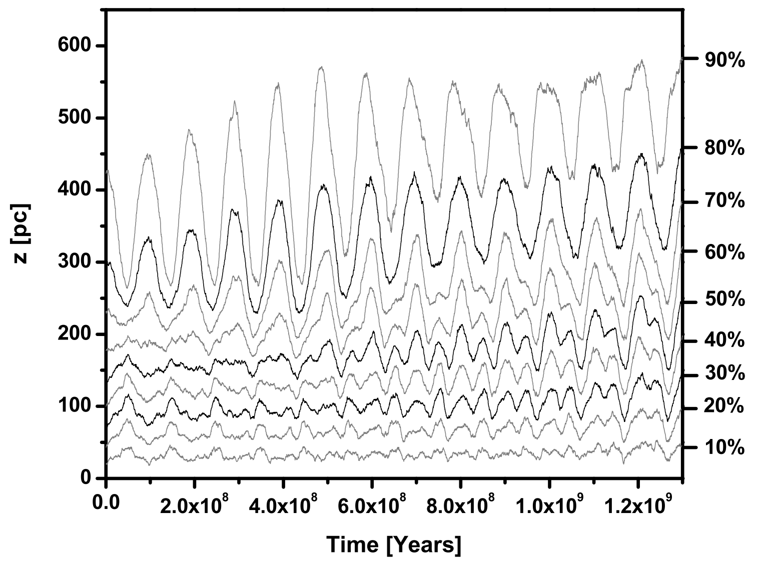

Figure 6.2: Development of a disc, that is being penetrated by a particle with $m = 20000\,M_0$. The external potential was chosen NOT to interact with the disturber.

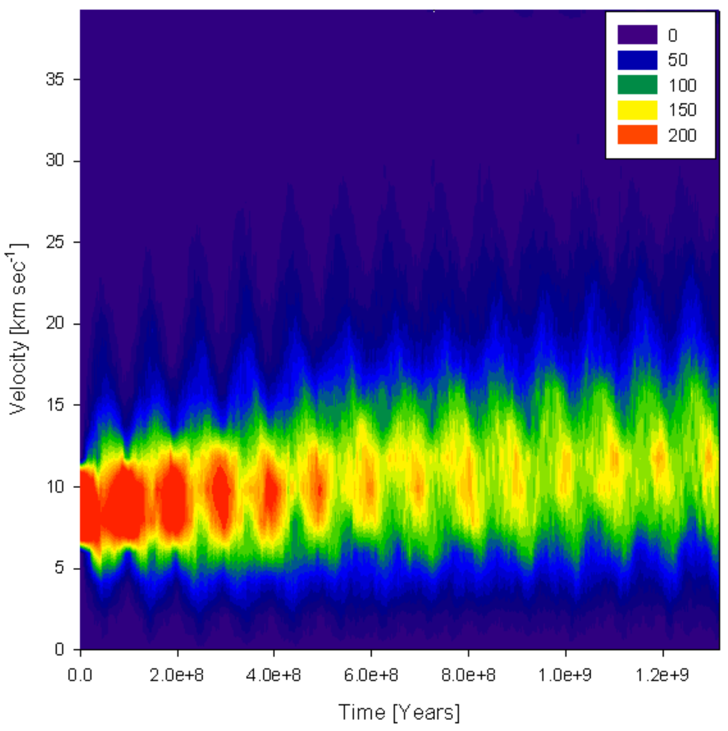

Figure 6.3: Statistical evolution of the velocity of the system. The number of particles in a specific interval is the colour coded value. The medium velocity has risen from $10\,km\,s^{-1}$ to about $13\,km\,s^{-1}$. Also the distribution of particles in velocity space has broadened.

6.2 An Unrealistic Disturber

The results of the simulation described in the last section led to the idea of excluding the effect of the external galactic potential on the disturber. The same setup has been used as before, except that the galactic potential was restrained from interacting with the disturber. The disturber was able to oscillate through the disc and was mirrored at $z = 0$, like all the other particles (corresponding to a particle being captured in the galactic potential), thus passing the same particles several times. This corresponds more or less to a continuous in fall of massive objects.

Looking at the stratification plot of this simulation in Figure 6.2, a clear expansion of the disc can be seen. The disturbing massive particle was now able to transfer energy to particles in the disc, resulting in an increase of temperature, thus being responsible for the expansion. The disc expanded from an initial scale height of $210\,pc$ to a scale height of about $300\,pc$. Further indications that the temperature of the disc has risen, can be obtained by analysing the kinetic energy data. These data show an increase in energy, meaning that particles have been accelerated. Again this corresponds to a rise in temperature.

A systematic acceleration of particles in the disc can be shown by evaluating the velocity data statistically. This has been done by analysing the velocity dispersion of the system using histograms. For each time of the simulation, a histogram with the interval of $2\,km\,s^{-1}$ has been created and joined together to an area plot, visualising the evolution of the system. The result can be seen in Figure 6.3 where an overall acceleration of the particles from $10\,km\,s^{-1}$ to about $13\,km\,s^{-1}$ can be detected. This acceleration is again an indication, that energy has been transfered from the disturber to the disc. Also a broadening of the distribution can be seen, again energy has been transfered.

7 Conclusion

It could be shown, that a massive object penetrating the galactic disc does indeed heat up the disc and can be made responsible for an expansion of the disc. A irregular potential does thus have an effect on the orbits of stars in the galaxy. However in this simulation, this effect could only be produced by making the unrealistic assumption, that the penetrating particle is not affected by the external galactic potential.

Since the disturber was able to oscillate through the disc, a scenario with a continuous in-fall of mass has been simulated. With these assumptions and an object with the mass of $2 \cdot 10^4\,M_0$, the galactic disc could be expanded from $210\,pc$ to $300\,pc$. The mean velocity increase was from $10\,km\,s^{-1}$ to $13\,km\,s^{-1}$. This is by far too little to be responsible for a heating up of the thin disc to the thick disc, because the end result should be $1300\,pc$ high, with a mean particle velocity of $40\,km\,s^{-1}$. This means that the effect of a penetrating massive object does not create the irregularities needed, to explain the creation of the thick disc.

Using objects with higher masses could lead to the desired effect. To study this, the simulation needs to be adopted considerably. A general problem with the disc heating theory is, that no observational data are known, showing an in-fall of massive objects exists. It is thus highly unrealistic, that the disc heating theory corresponds to reality. What is sure is, that objects passing through the disc do have an effect on the thickness, but only a very small one.

Acknowledgment

My profoundest thanks go to Gerhard Hensler for all the precious time he offered, without this bachelors thesis would not have been possible. It is not natural, that a

professor invests as many hours into a student, as Gerhard has done.

As well I would like to thank my father, who let me make his computer unusable for weeks and who did all the proofreading. Stellar physics is somewhat different to the physics of roads, so thanks for all the time trying to grasp all the aspects of galactic dynamics.

Appendix

A Detailed Calculations

A.1 Calculations in Section 2.4

The first taylor series derivative (2.12) can be found by keeping in mind, that

$$\frac{1}{R^3} = \frac{1}{\left(\sqrt{\vec{R}\cdot\vec{R}}\right)^3}. \tag{A.1}$$

The derivative of (A.1) is

$$\frac{d}{dt}\left(\frac{1}{R^3}\right) = -3\frac{1}{\left(\sqrt{\vec{R}\cdot\vec{R}}\right)^4}\cdot\frac{1}{2\sqrt{\vec{R}\cdot\vec{R}}}\cdot 2\vec{R}\cdot\frac{d}{dt}\vec{R} = -3\frac{\vec{R}\cdot\vec{V}}{R^5},$$

differentiation of (2.11) will lead to

$$\frac{d}{dt}\left(\frac{-m_j\vec{R}}{R^3}\right) = \frac{-m_j\frac{d}{dt}\vec{R}}{R^3} + 3\frac{\vec{R}\cdot\vec{V}}{R^2}\cdot\frac{m_j\vec{R}}{R^3}$$

which can be written as

$$\vec{F}_{ij}^{(1)} = \frac{-m_j\vec{V}}{R^3} - 3a\vec{F}_{ij}$$

with $a = \vec{R}\cdot\vec{V}/R^2$. Finding the next two Taylor series terms is quite lengthy, this is not shown here completely but some helping relations for this undertaking are:

$$\frac{d}{dt}\vec{V} = \left(\vec{F}_i - \vec{F}_j\right) \text{ and } \frac{d}{dt}\vec{F}_{ij} = \left(\vec{F}_i^{(1)} - \vec{F}_j^{(1)}\right)$$

$$\frac{d}{dt}a = \frac{d}{dt}\left(\frac{\vec{R}\cdot\vec{V}}{\left(\sqrt{\vec{R}\cdot\vec{R}}\right)^2}\right) = \frac{\vec{V}\cdot\vec{V}}{R^2} + \frac{\vec{R}\cdot\frac{d}{dt}\vec{V}}{R^2} - \frac{\left(\vec{R}\cdot\vec{V}\right)^2}{R^4}$$

$$b = \frac{d}{dt}a = \left(\frac{V}{R}\right)^2 + \frac{\vec{R}\cdot\left(\vec{F}_i - \vec{F}_j\right)}{R^2} + a^2$$

$$c = \frac{d}{dt}b = \frac{3\vec{V}\cdot\left(\vec{F}_i - \vec{F}_j\right)}{R^2} + \frac{\vec{R}\cdot\left(\vec{F}_i^{(1)} - \vec{F}_j^{(1)}\right)}{R^2} + a(3b - 4a^2)$$

Knowing this, gradually the second and third term can be found.

$$\vec{F}_{ij}^{(2)} = \frac{-m_j\left(\vec{F}_i - \vec{F}_j\right)}{R^3} - 6a\vec{F}_{ij}^{(1)} - 3b\vec{F}_{ij}$$
$$\vec{F}_{ij}^{(3)} = \frac{-m_j\left(\vec{F}_i^{(1)} - \vec{F}_j^{(1)}\right)}{R^3} - 9a\vec{F}_{ij}^{(2)} - 9b\vec{F}_{ij}^{(1)} - 3c\vec{F}_{ij}.$$

B Technical Details to the Implementation

My implementation of the NBODY0 source code provided by Binney & Tremaine in their Galactic Dynamics book, was written in Objective C / ANSI-C. The choice for this language has no particular reason, except that it is the standard language for the Apple OS X operating system. Using Objective C enabled the usage of object oriented programming, so that many parts of my code can be reused for other problems as well.

Since many mathematical operations used in the scheme are done using vectors, the usage of the PowerPCs AltiVec engine was quite obvious. AltiVec is a vector processing unit on the PowerPC G4 and G5 chip, enabling parallel processing of four numbers. This reduced many operations that required three operations in normal processing mode, into only one when using the AltiVec engine. A direct result of this is a speed increase of about a factor three. One drawback of AltiVec lies in number precision, since the best floating point type is only the ANSI-C type float (double float is not supported). Another drawback is the quite cryptic syntax, not enabling an intuitive way of writing equations. A quick introduction to AltiVec will be given below.

For visualization of the simulation, the visualization package VTK has been used. VTK is an open source package, mainly used in medical applications. It has a very steep learning curve and documentation is scarce. But the package offers many possibilities and is very powerful. For introductions on how to implement VTK with Apples OS X and especially the COCOA architecture, consult this very good tutorial: VTK Tutorial `http://www.macdevcenter.com/pub/a/mac/2003/02/11/dev_osx.html`

All the technical aspects in this Appendix are focused on the Apple OS X operating system. Although only the AltiVec part is mainly limited to the PowerPC architecture, all the other parts can be interesting for development on any platform.

C AltiVec

This section is meant to give a quick overview on how to use AltiVec with Objective C and Apples Xcode coding program. This is by far not a complete guide to programming with AltiVec and it will only cover aspects as far as they had been used in the simulation source code. For further and deeper information, the Apple developer homepage Delevoper Apple `http://developer.apple.com` is an essential place to look for.

To be able to use the AltiVec engine, the compiler needs to be informed, to use its extended functions for AltiVec. In Xcode this compiler flag can be easily turned on, by editing the Target properties. Using "Get Info" to edit the default Target in the project browser, AltiVec can be turned on under "Build", by activating the check box in front of "Enable AltiVec Extension". This will inform the gcc compiler to use the AltiVec extensions.

Declaration

Declaration of a variable with the type vector, is done as any other variable is defined and initialised in C. Vectors can be of different types, like character, integer or float. A complete list of all types can be found on Apples developer page. The following example will clarify how to initialise a vector:

```
vector float someVector;
vector int someIntVector;
someVector = (vector float) (1.23, 0.0, -1.85, 54.34);
someIntVector = (vector int) (1, 4, 34, -4);
```

Basic Math

Doing math with vectors is not straight forward with AltiVec and shows similarities to the Assembler programming language. This makes AltiVec code very hard to read. In the source code, I tried to comment the underlying calculation, so it should be possible to see how things work. A short introduction to the most basic operations shall be given. With these, more complicated structures can be build.

Unlike doing normal math in C, AltiVec uses functions to do the basic operations. This means for example, that adding just two vectors is not done in the intuitive way of c = a + b. Instead one has to use c = vec_add(a, b). The following table describes the frequently used operations.

Function	Syntax	Description
vec_add	vec_add(arg1, arg2)	Adds two vectors together. $\vec{arg}1 + \vec{arg}2$
vec_sub	vec_sub(arg1, arg2)	Subtracts two vectors. $\vec{arg}1 - \vec{arg}2$
vec_madd	vec_madd(arg1, arg2, arg3)	Multiplies each component of arg1 and arg2, and adds this to arg3. $arg1_i \cdot arg2_i + arg3_i$
vec_nmsub	vec_nmsub(arg1, arg2, arg3)	Subtracts the component wise product of arg1 and arg2, from arg3 $-(arg1_i \cdot arg2_i - arg3_i) = arg3_i - arg1_i \cdot arg2_i$

Extended functions - vecMath.h

Some additional vector functions are included in vecMath.h. There are two types of functions in this library. Some are implementations of some extended vector operations using AltiVec, other can be used to store and retrieve information in vectors.

The extended vector operations implemented in vecMath.h are the scalar product, absolute value of a vector, and the square absolute value.

Function	Syntax	Description
vecSdot	vecSdot(arg1, arg2)	Returns the scalar multiplication of arg1 and arg2 as type float
vecAbs	vecAbs(arg1)	Returns the absolute value of arg1 as type float
vecAbsQuad	vecAbsQuad(arg1)	Returns the squared value of vecAbs as type float

The following functions in vecMath.h can be used to manipulate data stored in vectors directly. They also make it possible, to extract only certain components of the vector, all functions that are not available by default.

Function	Description
createUniformVector(float x)	Returns an vector that has the same value x in every component. Used to multiply a vector with a "scalar"
convertVectorToArray(arg1, float(*array))	Writes the components of an vector into an existing array of the type float.
createVector(x, y, z)	Used to dynamically create vectors.

D VTK

VTK is an open source visualization library, used to visualize many problems in different kinds of scientific disciplines. It can be downloaded on the Kitware homepage VTK `http://public.kitware.com/VTK/` or on OS X using fink.

VTK is written in C++, so applications using VTK need to be developed in C++ too. This is no particular drawback, since the Apple gcc compiler allows to mix the Objective C and C++ languages. To get started with VTK is quite painfull, since the documentation is quite sparce for beginers. An easy way to start is, to look at the tutorial for VTK on Mac VTK Tutorial `http://www.macdevcenter.com/pub/a/mac/2003/02/11/dev_osx.html`. Then there are many good example programs that try to show how specific aspects of VTK are used. These can be found on the VTK documentation pages VTK Doc `http://public.kitware.com/VTK/doc/nightly/html/pages.html`. A good thing is also to look at my visualisation program used for generating movies of my N-Body simulations, that is a crude adaption of the Tutorial mentioned above.

E Class Structure and Description of the Source Code

In this section, the source code with all its classes will be discussed. The source code can be downloaded on my homepage Icysun `http://cs.icysun.net`. It is being distributed under the GPL license.

Five classes make up the whole simulation program, of which one is responsible for the setup of the particle system to be simulated.

The classes `body` and `particleSystem` make up the system to be simulated. The class `body` defines the a particle of a system, with all its properties and methods. The class `particleSystem` gathers many bodies into one system. `particleSystem` defines all the properties and methods of a particle system.

The class `setupSystem` will produce a particle system, that is either homogenically distributed or a part of the galactic disc with all the properties described above.

The `integration` class handles all the math and is the main part of the simulation code. A `particleSystem` will be passed to this class and is then simulated.

The main class `nBodyBac2` is handling the program flow and is responsible for the main menu.

`body`

The `body` class defines one particle of the particle system. It defines methods of reading and writing all the particle properties.

Property	Type	Writing Method	Reading Method
idNum	int	setIdNum	getIdNum
mass	float	setMass	getMass
r_0	vector float	setR_0	getR_0
r_t	vector float	setR_t	getR_t
v_0	vector float	setV_0	getV_0
v_t	vector float	setV_t	getV_t
F	vector float	setF	getF
F1	vector float	setF1	getF1
F2	vector float	setF2	getF2
F3	vector float	setF3	getF3
D1	vector float	setD1	getD1
D2	vector float	setD2	getD2
D3	vector float	setD3	getD3
delta_t	double	setDelta_t	getDelta_t
t_0	double	setT_0	getT_0
t_1	double	setT_1	getT_1
t_2	double	setT_2	getT_2
t_3	double	setT_3	getT_3
hasPotentialON	int	setReactOnPotentialON setReactOnPotentialOFF	reactsOnPotential

Initialisation is being done with `[body initWithId:(int) newID withMass:(float) newMass withR_0: (vector float) newR_0 withV_0: (vector float) newV_0]`

particleSystem

The `particleSystem` class manages the particle system that needs to be simulated. It gathers many bodies into one system an handles general management methods for the particles. `particleSystem` has the following properties:

Property	Type	Writing Method	Reading Method
* particle	NSMutableArray	see below	getParticle
numParticle	int		count
boxDimX	float	setBoxDim	getBoxDimX
boxDimY	float	setBoxDim	getBoxDimY
boxDimZ	float	setBoxDim	

The `NSMutableArray * particle` holds all the bodies in the particle system and is of the type NSMutableArray, as used in the COCOA environment. To add a particle, the following three methods can be used:

- `[particleSystem addParticle: (body *) body]` - this will add an already initialised object of the type body to the array of particles.

- `[particleSystem addParticleWithMass: (double) newMass withR_0: (vector float) newR_0 withV_0: (vector float) newV_0]` - this will add a new body with the properties mass, R_0 and V_0 to the array of particles. This body will interact with an external potential.

- `[particleSystem addParticleWithMassPotentialOFF: (double) newMass withR_0: (vector float) newR_0 withV_0: (vector float) newV_0]` - the same as `addParticleWithMass`, except that this body will be excluded from interaction with an external potential.

Two other methods are included in `particleSystem` for body management. One is to replace an existing body with a new one and the other method will delete an existing body.

- `[particleSystem replaceParticleWith: (body *) tmpBody atIndex: (int) idNum]`
- `[particleSystem deleteParticleAt: (int) idNum]`

`integration`

The `integration` class implements the integration scheme as described above. It also controls its own program flow and once started it cannot be modified from outside of the class.

But before the integration itself is invoked, the class needs to be setup correctly, no check is being carried out, if it has been setup correctly! All the parameters need to be passed on. This is done by the `setupSystem` class that will guide the user through the setup process. To see what need to be configured, it is good to look through the `setupSystem` class.

`integration` has the following properties:

Property	Type	Writing Method	Reading Method
epsilon	float	setEpsilon	getEpsilon
currentTime	double	setCurrentTime	getCurrentTime
startingTime	float	setStartingTime	getStartingTime
criticalTime	float	setCriticalTime	
eta	double	setAccuracy	getAccuracy
withExternalPotential	int	turnExternalPotentialON	
		turnExternalPotentialOFF	
potentialDistance	float	setPotentialDistance	
externalPotentialMass	float	setPotentialMass	
mirrorOnZ	int	turnMirroringZON	
		turnMirroringZOFF	
keepBoxSizeFix	int	turnReturnToBoxON	
		turnReturnToBoxOFF	
deltaT	float	setOutputInterval	

To initialise the integration class, the following order of method-calls needs to be done:

```
[integration initialisePartOne];
[integration initialisePartTwo];
[integration initialiseTimeSteps];
[integration integrate];
```

F Appendix: Some small notes

F.1 External Potential

When using a spherical potential as an external potential, it is best to use a plummer sphere with a=0. This results in a normal sphere. The advantage of this is, that the plummer potential has a Taylor series expansion that is similar to the one of the gravitational force. The terms that need to be added to the force polynom are:

$$\vec{F_P} = \frac{M\vec{r_i}}{\left(r_{ij}^2 + a^2\right)^{3/2}}$$

$$\vec{F_P^{(1)}} = \frac{M\vec{v_i}}{\left(r_{ij}^2 + a^2\right)^{3/2}} - 3\frac{\vec{r_i}\cdot\vec{v_i}}{\left(r_{ij}^2 + a^2\right)^{3/2}} \cdot \vec{F_P}$$

Be sure to double check the position in the source code of where the external potential is added to the force. Be careful not to place it in the loop, where the influence of all particles on the one particle is evaluated. It should be added before or after this loop, not inside!

F.2 Calculation of Energy

If the kinetic energy shows in its plot signatures that should not be there, check for particles that escaped into the maxwell tail. A good way to calculate the kinetic energy is, to filter the data for these kind of particles (maybe using some kind of sigma clipping?). This filtering mechanism has not been implemented in the analysis routines.

F.3 Simulating Very Heavy Particles

Is the disturbance particle very heavy (definitely greater than 20000), then this particle should be given a different value of ϵ. This needs to be a special feature of the particle and has to be implemented into the particle class. Integration code has thus to be adopted. This will prevent the heavy particle from swallowing up all the other particles and colliding with them. The heavy particle should have a softer collision than all the others.

F.4 Apples Transition to Intel Processors

The complete code needs to be rewritten, so that it could be reused on the next to come Intel architecture. For this it would be best, to adapt the whole code for use of the vecLib.framework (i.e. vDSP). This would enable easy cross platform usage of the code. Again, the AltiVec advantage can only be leveraged, with the usage of single precision floats, even though, vecLib.framework allows the processing of double precision floats (they are processed using scalar code).

References

Aarseth, S. J. 2001, New Astronomy, 6, 227

—. 2003, Gravitational N-Body Simulations (Cambridge University Press)

Binney, J. & Tremaine, S. 3. edition, 1994, Galactic Dynamics (Princeton Series in Astrophysics)

Buser, R. 2000, Science, 287

Font, A. S., Navarro, J. F., Stadel, J., & Quinn, T. 2001, Astrophysical Journal, 563, L1

Freeman, K. & Blend-Hawthorn, J. 2002, Annu. Rev. Astron. Astrophys.

Gilmore, G. 2003, RevMexAA, 17, 149

Gilmore, G. & Reid, N. 1983, Mon. Not. R. astr. Soc.

Gilmore, G., Wyse, R., & Kuijken, K. 1989, Annu. Rev. Astron. Astrophys.

Makino, J. 1991, Astrophysical Journal, 369, 200

Nordström, B., Mayor, M., Andersen, J., Holmberg, J., Pont, F., Jorgensen, B. R., Olsen, E. H., Udry, S., & Mowlavi, N. 2004, A&A

Wielen, R. 1977, Astron. Astrophys.

Wielen, R., Dettbarn, C., Fuchs, B., Jahreiss, H., & Radons, G. 1992, The Stellar Populations of Galaxies, 81