Is mass segregation in star clusters primordial? Simulations of spherical star clusters

Martijn van Pottelberghe
Onne Slooten
Welmoet Damsma

17th June 2003

With many thanks to our supervisor
Simon Portegies Zwart

Abstract

We have written a program in C to investigate the primordiality of mass segregation in star clusters. As integration method we use the time-symmetric leapfrog. From the simulation data we calculate $\ln \Lambda$ (the coulomb logarithm). The method we use for this is not very precise, but when we compare our result with the $\ln \Lambda$ Bonnell and Davies have found in a similar study in 1998 [1], we find that they are almost exactly the same. From our results we can conclude that mass segregation takes place and from the comparison with Bonnell and Davies we find that it may be primordial.

Contents

1 Introduction

In spherical star clusters all stars are born at the same time. In recent years observations with advanced telescopes have shown that massive stars in star clusters are typically found in the cluster centre. The question is: are these stars born in the centre or are they formed all over the cluster and do they sink towards the centre in the course of time? Such sinking can be achieved by gravity and dynamical friction. In other words, is mass segregation primordial or not? In 1998 Bonnel and Davies wrote an article about mass segregation in young star clusters [1]. They concluded that the massive stars must have formed near or in the centre of the cluster. They performed a numerical N-body simulation to investigate the time-scale necessary for the dynamical mass-segregation to occur. The program they use is the NBODY2 code, developed by Sverre Aarseth [2]. The code uses a softening factor to model the interaction between galaxies. Aarseth states about NBODY2: "However, it does not contain any special treatments of close encounters and should therefor only be used for problems where the effect of hard binaries can be neglected[...]." [3] The young star clusters Bonnel and Davies studied do have close encounters. Therefore we would like to do the same thing, but using an appropriate program. We mean to write this code ourselves. Furthermore, we are going to investigate the theoretical background of mass segregation (in Section 2). In section 3 we elaborate on our method of working and present our results. The last section consists of a discussion of our method and results and we draw our conclusions.

2 Theory

It is unknown whether star clusters are formed mass-segregated, or evolve later on to a mass-segregated state. Considering the last option, possibly, this evolution could be due to the following effect: When two bodies, one more massive than the other, get close to each other, they will feel each others gravity. The gravitational pull of the massive body will increase the kinetic energy of the less massive body. This energy is taken from the potential energy of the massive body. This is called a two-body encounter.
Encounters like this occur frequently in dense star clusters. As a result of them, the kinetic energy of the most massive stars will drop continuously and they will sink to the bottom of the potential well of the star cluster. On the other hand, very light stars gain kinetic energy with every two-body encounter. Some of them may even escape from the potential well entirely, however, this only happens to extremely light stars, which do not contribute very much to the mass-function of the star cluster. Therefore, we do not consider this effect in the theoretical part if our investigation.

We investigate if two-body encounters provide a good explanation for the observed mass-segregation. To do this, we determine how much time it takes to form a mass-segregated cluster. We start with the density as a function of radius according to a plummer-sphere:

$$\rho = \frac{3M_0}{4\pi r_c^3} \frac{1}{(1 + r^2/r_c^2)^{(\frac{5}{2})}} \tag{1}$$

In this equation r is the distance from star to cluster center, r_c is the core radius, which defines the plummer sphere and M_0 is the total mass of the cluster. Integrating this formula gives us the mass distribution:

$$M(r) = \frac{M_0 r^3}{(r^2 + r_c^2)^{(3/2)}} \tag{2}$$

From this we can derive expressions for circular velocity, acceleration and potential:

$$v_c^2 = \frac{GM(r)}{r} = \frac{GM_0 r^2}{(r^2 + r_c^2)^{(3/2)}} \tag{3}$$

$$a(r) = \frac{GM(r)}{r^2} = \frac{GM_0 r}{(r^2 + r_c^2)^{(3/2)}} \tag{4}$$

$$\phi(r) = -\int a(r)dr = \frac{GM}{\sqrt{r^2 + r_c^2}} \tag{5}$$

By adding kinetic and potential energy (both divided by m), we find an expression for the total energy (divided by m) of a circular orbit:

$$E_c = \frac{1}{2}v_c^2 + \phi = \frac{GM(3r^2 + 2r_c^2)}{2(r^2 + r_c^2)^{(3/2)}} \tag{6}$$

The time-derivative of this equation is:

$$\frac{dE_c}{dt} = -\frac{3GMr^3)}{2(r^2 + r_c^2)^{(5/2)}} \frac{dr}{dt} \tag{7}$$

Another way of getting this, is by taking the acceleration for dynamical friction [4]:

$$\mathbf{a}_f = -4\pi \ln \Lambda G^2 \rho m \frac{\mathbf{v}_c}{v_c^3} \chi \tag{8}$$

The important factors in this expression are m, which is the mass of the star, and $ln\Lambda$, which is the coulomb logarithm. Substitution gives us:

$$\frac{dE_c}{dt} = -\chi \ln \Lambda G^2 \frac{\rho m}{v_c} \tag{9}$$

Substituting equation (1) and (3) gives:

$$\frac{dEc}{dt} = -\chi \ln \Lambda G m \frac{3\sqrt{GM}}{4\pi r r_c^{(3/2)}} (1 + \frac{r^2}{r_c^2})^{(-7/4)} \tag{10}$$

If we equate these two expressions for the time derivative and solve for dr/dt we get:

$$\frac{dr}{dt} = -\chi \ln \Lambda \sqrt{\frac{G}{M}} \frac{(1 + \frac{r^2}{r_c^2})^{(3/4)} r_c^{(7/2)}}{2\pi r^4} \tag{11}$$

We do not solve this differential equation analytically. Instead, we use numerical integration. For this we use a standard trapezium integration code (the program is called trapezium.c, see Appendix H). We also use the initial condition $r(t = 0) = r_0$. This gives us the timescales as shown in the results section (section 3.5).

3 Experimental Procedures

To simulate a young spherical star cluster we write a N-body integration program in C. We use a straightforward method: the particle-particle method.

3.1 The program

We created a collection of programs. They are listed here:

[**incon**]

This program creates initial conditions for the cluster, using the Plummer-sphere model (see section 3.5.1 and appendix A for the code).

[**starsim**]

This program evolves the cluster, using the leapfrog method (section 3.2). It is basic; it uses a constant timestep that can be set. You can tell how many timesteps are to be calculated. If the cluster is not in virial equilibrium, starsim can also normalize it (see appendix B for the code).

[**mathout**]

Used to visualize the data-output from starsim by way of a mathematica notebook (see appendix C).

[**gnuview**]

Used to visualize the data-output from starsim using gnuplot (see appendix D).

[**clscat**]

Used to merge two data-outputs from starsim into one file (see appendix E).

[**inconfromcls**]

Used to take the last timestep's positions, masses and velocities from a starsim-data-output, and forward it as "initial conditions" to starsim again, so it can calculate further in time, taking the result from a previous run as a basis. The result of this starsim-run will lack the part of the simulation-data-output up to the time where starsim had stopped the previous time. The two parts can then be concatenated into one file again using clscat (see appendix F).

[**getinit**]

Retrieves from a starsim-data-output the initial conditions for each star (see appendix G).

3.2 Integration method

The main program, starsim, integrates the paths of all stars using the leapfrog method. We use a time-symmetrized version of the code, in order to better conserve the total energy of the system. What we explain here is a non-time-symmetrized version, to get the idea.

For each timestep, the acceleration is calculated for each star using Newton's law of gravitation ($F = -\frac{GM1M2}{r^2}$). This is what takes most of the time: The number of forces that has to be calculated is proportional to the square of the number of stars you are simulating.

One integration step consists of calculating the new positions of the stars, then calculating the new velocities.

The new positions are calculated by $r_1 = r_0 + v\ \delta t$. where $r_0 = r(t)$ and $r_1 = r(t + \delta t)$. But what v should be used?

See Figure 1. One can easily see, that neither the velocity at time $t(v_0)$, nor the velocity at time $t + \delta t(v_1)$ should be used, but something in between (the blue line in the picture). We just take the average of v_0 and v_1: $\frac{v_0+v_1}{2}$. This we call $v_{\frac{1}{2}}$.
To sample the velocities at times that lie in between the times where the positions are sampled, in the first step you will have to calculate $v(\frac{\delta t}{2})$. We approximate this by $v(\frac{\delta t}{2}) = v(t = 0) + \frac{1}{2}a(t = 0)\ \delta t$

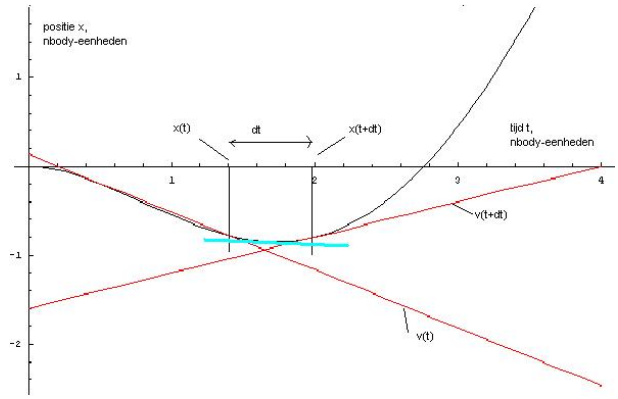

Figure 1: Explaining the leapfrog

Each integration step then consists of

$$r_1 = r_0 + v_{\frac{1}{2}}\ \delta t \tag{12}$$

$$v_{\frac{3}{2}} = v_{\frac{1}{2}} + a_1\ \delta t, (v_{\frac{3}{2}} = v(t + \frac{3}{2}), v_{\frac{1}{2}} = v(t + \frac{1}{2})) \tag{13}$$

For the time-symmetrized version of the leapfrog, the equations become:

$$r_1 = r_0 + \frac{1}{2}(v_1 + v_0)\delta t + \frac{1}{12}(a_1 - a_0)(\delta t)^2 \tag{14}$$

$$v_1 = v_0 + \frac{1}{2}(a_1 + a_0)\delta t + \frac{1}{12}(j_1 - j_0)(\delta t)^2 \tag{15}$$

Where the jerk $j = \frac{da}{dt}$ (calculated by differentiating the expression for the force).
For a full discussion of the leapfrog, it's time-symmetrized version and it's 4th order versions, read the article "Building a better leapfrog" by Piet Hut, Jun Makino en Steve McMillan [8].
See Appendix B for the code.

3.3 Testing the program

It is difficult to determine whether the program works properly. The results must be physically plausible, but this is not always easy to determine. To test the program test initial conditions can be used, the outcome of which are to be expected. We use the Montgomery problem and the Pythagoras problem to test our code (see Appendix D for the codes).

3.3.1 Montgomery problem

The montgomery problem is not a difficult one to integrate. There are three stars that circle around each other very regularly. There are no close encounters between the stars. This test is a good means to see whether the integration code does what it should. Starsim runs the test well. Even after 100 n-body time units, for a timestep of 10^{-4}, the simulation was rock-solid.

3.3.2 Pythagoras problem

This test is really good to determine whether the precision of the integration-code is problematic. Three stars with different masses start out at rest, at specific positions. Then they fall to each other, and come really close to each others. This, starsim did not do as well as the montgomery problem; the system 'exploded' after about 15 n-body time unit (for a timestep of 10^{-4}).

3.4 Units

To simplify matters and save computing time, it is convenient to use a different unit system. These are the so-called Standard N-body Units [9]. In this system the total initial mass and the gravitational constant G equal 1. The total initial energy E equals $-\frac{1}{4}$. This is convenient, because we do not have a specific initial energy and mass. We also take the core radius to be 1 (this is the radius in which 50 percent of the mass is located).

3.5 Results

3.5.1 Initial conditions

Plummer's model (see section 2) is used to set the initial conditions. For this, we follow the procedure described in the appendix of the article *"A Comparison of Numerical Methods for the Study of Star Cluster Dynamics"* by Aarseth et al [6]. This procedure produces random initial conditions in Standard N-Body

Units (section 3.4) for a Plummer sphere. Each body is characterized by 7 variables: its position r (x,y,z), the velocity v (vx,vy,vz) and its mass m. We do not take into account stellar evolution. The position and the velocity are updated when simulating the interaction.
The cluster mass distribution is either all masses are the same, or ten of them are heavier than the others (by a factor of 10 or 20), in order to study the effects of mass segregation.
To generate random numbers we use a code taken from the internet, developed by Steve Park [7].
See Appendix A for the code.

3.5.2 The simulation

We simulate a cluster of a 1000 stars, of which 10 are of mass $M_{massive}$. The rest is of mass $\sim \frac{1}{1000}$. Total mass is 1. We performe several runs, to test the program and run out statistical errors.

Run	N_{total}	$M_{massive}$	Timestep	Time from t=0 to t=
1	1000	0.01	$1 \cdot 10^{-4}$	20
2	1000	0.01	$1 \cdot 10^{-4}$	20
3	1000	0.02	$1 \cdot 10^{-4}$	20
4	1000	0.02	$1 \cdot 10^{-4}$	20
5	1000	0.02	$1 \cdot 10^{-5}$	10
6	1000	0.02	$2 \cdot 10^{-3}$	400
7	1000	0.02	$2 \cdot 10^{-3}$	80
8	1000	0.02	$5 \cdot 10^{-5}$	2
9	1000	0.02	$1 \cdot 10^{-4}$	20
10	1000	0.01	$1 \cdot 10^{-4}$	20

Run number 1 was done to t=10, stopped and then continued to t=20. This caused an abrupt transition in the curve for the paths of the heavy stars. The data are still usable until that point (at t=10)
Runs number 6 and 7 are not usable, because the total energy is not constant from the beginning. This is probably caused by the large timestep.
Run number 8 is too short to use the data.
Runs number 2, 3, 4, 5, 9 and 10 all produced good data.

3.5.3 Graphs

The total energy of the star cluster should be constant in time. The figures 2 to 5 show the course of the total energy in time for runs number 3, 5, 7 and 10 respectively (see section 3.5.2). The total energy is stable up to a certain point. We have to chop off our data from that point. For run number 7 (figure 2, graph 3) the energy is not stable from the beginning, we do not use these data at all.

Theoretically, the path of a heavy star is a smooth curve, as shown in figure 5. The star first moves slowly towards the middle of the star cluster, speeds up and moves fast into the centre. From the simulation data we see that in reality the path winds and twists. The theoretical path can be visualized as a line through the average of the twists. For heavy stars from our simulation data we have drawn this line and guessed the continuation of the path of the stars. We print the paths of the heavy stars and draw through this, by hand, the theoretical line and extrapolate. This results in a time in which the star in our simulation should have reached the centre of the star cluster if we would have been able to simulate all the way. These are the dots in figure 6. Through the acquired data we fit a line (the continuous line in figure 6). There are 6 stars starting at R=1. From this and the fitted line we guess that our error bar is approximately 10 N-body units. The stars with a starting radius of R=0.5 or smaller are within the error bar, but all above the fitted line. These stars are already very close to the centre of the star cluster in the beginning. Therefore, we expect them to behave differently. The star at R=2 is outside the error bar. The path of this star is shown in figure 5. As can be seen the path is very eccentric and maybe this causes the strange behavior of this star. In a very eccentric orbit, the star almost immediately comes very close to the cluster centre and as with the stars that are already close to the centre at the start, the behavior is not what is theoretically expected.

The fitted line is constructed with the trapezium code, which was made to model the theoretical path of the heavy stars. The theoretical graph (the dotted line in figure 6) and the fit (the continuous line in figure 6) are the same, only the coulomb logarithm $\ln \Lambda$ is different. In theory, $\ln \Lambda$ is taken 0.1 N, with N the number of stars in the cluster (this is a number taken from literature). This makes in our case $\ln \Lambda$ 100 and Λ 4.6. This is the dotted line. For the continuous line we have taken $\ln \Lambda = 1.63 \cdot 10^5 = 163N$, what makes $\Lambda = 12$. This result is found by comparing the values of R=1 for from the theoretical graph and the simulated dots. The average of the dots for R=1 gives us a $t \approx 26$. The theoretical graph gives for R=1, $t \approx 10$. This differs a factor 2.6. Since $\Lambda = 4.6$ theoretically, for our simulated data it must be $\Lambda = 2.6 \cdot 4.6 = 11.96 \approx 12$. We filled in this value in the trapezium code and got a line that fits the dots nicely (the continuous line in figure 6).

Bonnell and Davies [1] did a similar research project, but have a different method. When corrected for a

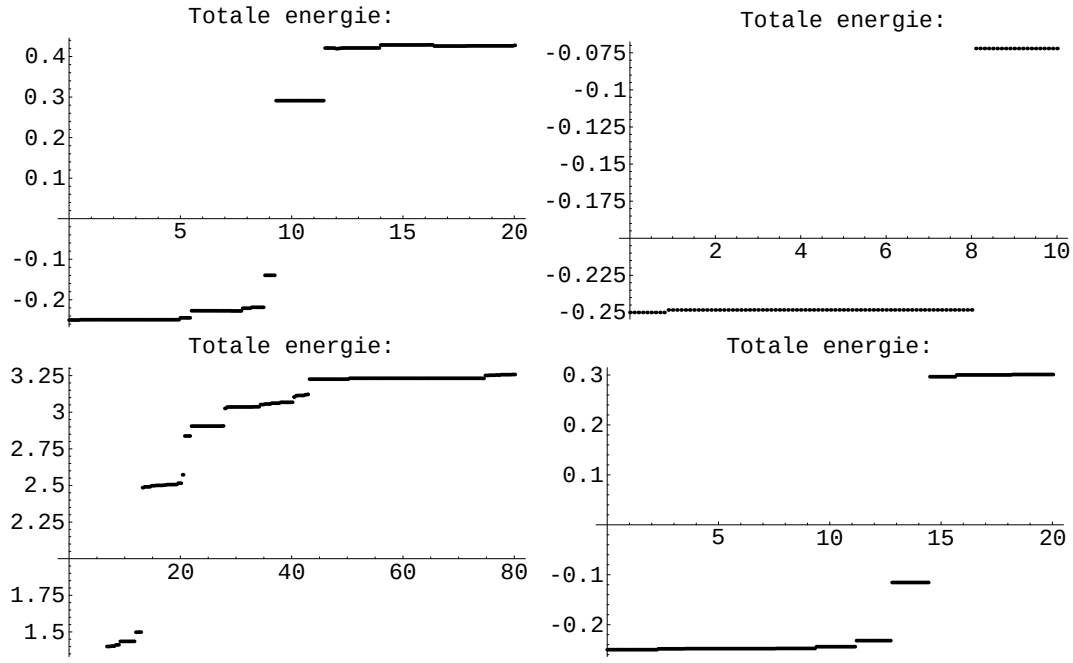

Figure 2: The total energy of the star cluster in runs number 3, 5, 7 and 10 as a function of time (N-body units).

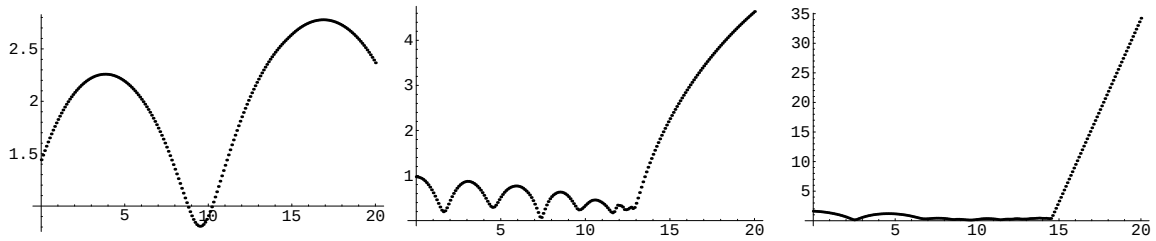

Figure 3: Paths of heavy stars in run number 10 (t on the horizontal axis and r on the vertical axis).

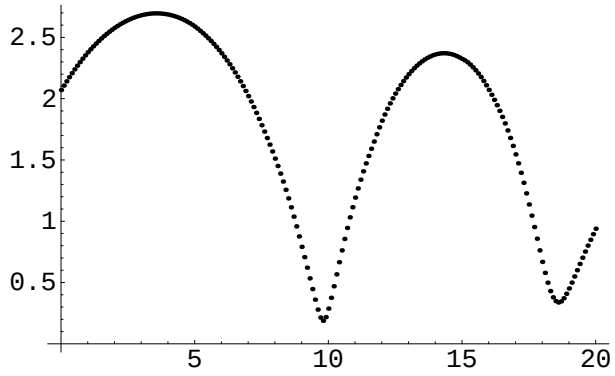

Figure 4: Path of the massive star, starting at R=2 in run number 4 (with t and r respectively on the horizontal and vertical axis).

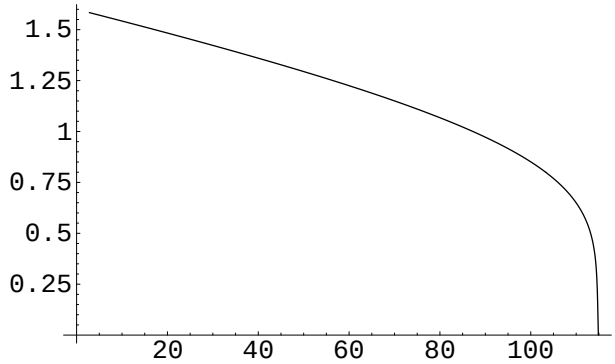

Figure 5: Theoretical path of a heavy star, starting at r=1.6 (t and r respectively on the horizontal and vertical axis).

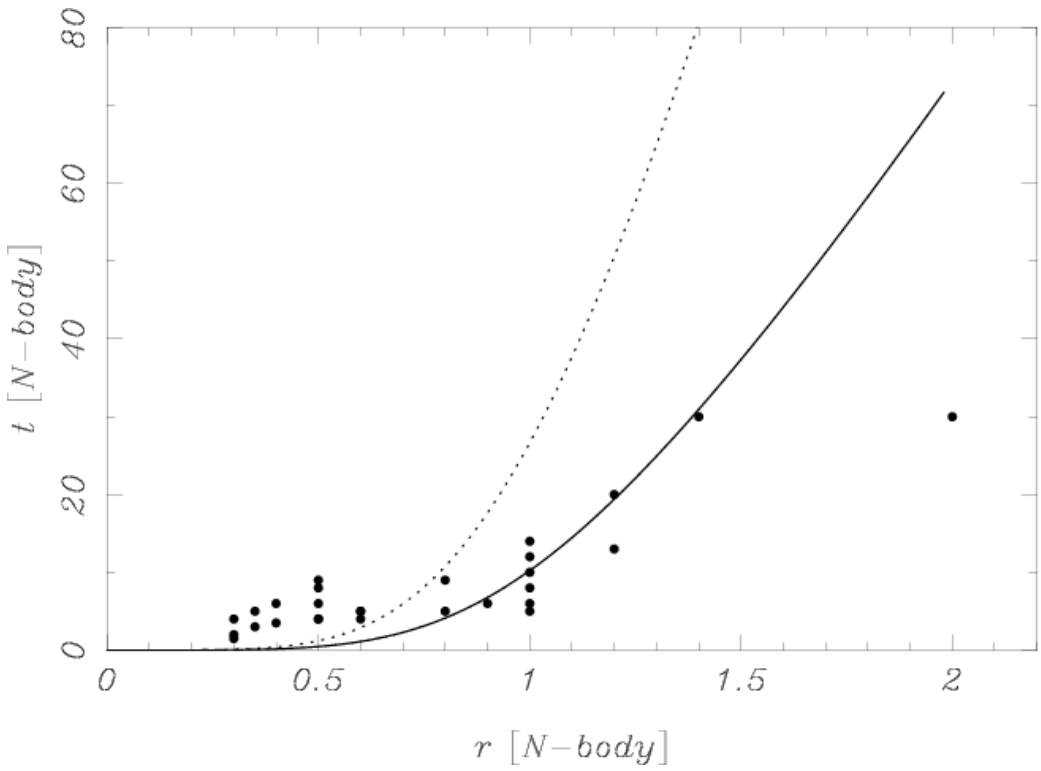

Figure 6: A star starting at r reaches the centre of the star cluster in t. The dotted line is the theoretical graph. The dots represent single massive stars (0.02 times the mass of the light stars), the data extracted from the graphs of the paths of the massive stars (for example as in figure 3). The continuous line fits the dots.

different mass-distribution and (...), our $\Lambda = 12$ appears to be consistent with their Λ. This can be calculated from figure 3 from their article, printed in Appendix J. Since they compared this result with observational data, we are also able to say something about whether mass segregation is primordial or not.

4 Discussion & Conclusions

4.1 Discussion

One of the biggest problems in this project was computer time. Particle-particle methods have a high computational cost, in the order of N^2, where N is the number of stars. It limits the number of stars and the number of timesteps we can take, but for our goal we needed no more than about a 1000 stars. The runs described in section 3.5.2 each took about 5 to 20 hours.

The result of the simulation is very dependent on how precise the program works. A good integration method is required. The time-symmetric leapfrog integration scheme is widely used for this kind of problems (involving second order problems). The leapfrog method is only a second-order scheme, but very stable and precise (compared with for example Runge-Kutta methods). It is also easy to use and code. Because the leapfrog is time reversible (it is symmetric), it conserves energy and angular momentum, which is highly desirable (as also discussed in section 3.4.3).

The program does seem to work very well up to a certain point. From that point on the total energy is not stable anymore. This point lies between t=5 and t=20 for a timestep between 10^{-4} and 10^{-5}. From that point on we find the data unreliable. What happens there might be the result of a very close encounter between two or more heavy stars in the centre of the star cluster. Our program cannot handle the very large numbers such a close encounter produces. This is illustrated in figures 7 and 8, where you can see that two heavy stars are shot out of the cluster. This happens almost exactly at the transitions of the energy in the same run, as can be seen in figure 2. We can use only the first range of data. Comparison with the theory and drawing conclusions from such a small range is difficult. The theoretical curve is similar to the simulation curve (see figures ...). The theoretical curve is smooth and runs from t=0 to t=... To compare the two, one must draw a smooth line through the data points of the simulated path of a heavy star (for example the path in figure 3). We do not have a precise method for this. We have only a very small part of the supposed path of the star. To draw a line is to guess a line. Therefore, the $\ln \Lambda$ we find is not very reliable. But Bonnell and Davies find the same value for the same kind of situation. They have studied observational results (for example the trapezium stars in the Orion nebula cluster) to compare them with the research results. From this, they conclude that mass segregation is primordial. If our value of $\ln \Lambda$ is reliable enough, we can say that our results support that conclusion.

4.2 Conclusions and suggestions

From our simulation data we can conclude that the massive stars do move towards the middle of the star cluster. From extrapolating the data we find the timescale on which the massive star should reach the cluster centre. From this we find a $\ln \Lambda$ that is consistent with the results of Bonnel and Davies [1]. They conclude that mass segregation is primordial. Our study supports their conclusion. But our data are not very reliable and the way we extracted $\ln \Lambda$ is not very precise.

The program is apparently not good enough to do this kind of simulations. Perhaps it might have been better to force the heavy stars in a circular path at radius 1. The big advantage of this is that one does not have to reckon with the eccentricity of the paths of the stars. It makes it easier to draw a line through the simulation data of the path of the star. Maybe it would have been better to have used only one heavy star, to prevent collisions between stars taking place. But then it would take too long to get a considerable amount of data. In principle the heavy stars are not bothered by one another, but maybe the energy would have stayed stable longer. With the Pythagoras problem we already discovered that our program does not like close encounters.

References

[1] I.A. Bonnel, M.B. Davies. *Mass Segregation in Young Stellar Clusters*, Mon. Rot. R. AStron. Soc. 295, 691 - 698 (1998).

[2] S.J. Aarseth, 1985.

[3] S.J. Aarseth. ....., Chapter 9: Small N-Systems
?? Gravitational N-Body SimulationsTools and Algorithms (1985?)

[4] J. Binney, S. Tremaine. *Galactic Dynamics* (page 425), Princeton: Princeton University Press (1987).

[5] H. Mouri, Y. Taniguchi. *Mass Segregation in Star Clusters: Analytic Estimation of the Timescale*, The Astrophysical Journal 580, 844-849 (2002).

[6] S.J. Aarseth, M. Henon, R. Wielen. *A Comparison of Numerical Methods for the Study of Star Cluster Dynamics*, Astron. & Astrophysics 37, 183 - 187 (1974).

[7] Steve Park, Random Number Generator: http://www.cs.wm.edu/ va/software/park/park.html

[8] P. Hut, J. Makino, S. McMillan. *Building a better leapfrog*, The Astrophysical Journal 443, L93-L96 (1995).

[9] D.C. Heggie, R.D. Matthieu. *Standarised Units and Time Scales* from *The Use of Supercomputers in Stellar Dynamics*, Springer-Veralg (1986)

[10] Source of articles on the internet: The NASA Astrophysics Data System, *http* : *//adsabs.harvard.edu/physics_service.html*

A incon.c

```
#include <stdio.h> #include <math.h> #include "rngs.h"

#define pi acos(-1.)

double g (double) ;

int main(int argc, char **argv) {
    int heavystars;
    int i;
    double r,M ;
    double x,y,z ;
    double vx,vy,vz ;
    double X1,X2,X3,X4,X5,X6,X7 ;
    double V, Ve, q ;
    int nstar=10;

    PlantSeeds(-1); /** [martijn]: random seed wordt uit
    systeemklok gedestilleerd **/

    if(argc==1) fprintf(stderr,"geen aantal opgegeven;
    ik ga uit van 10 sterren\n");
    else if(argc==2||argc==3) sscanf(argv[1],"%d",&nstar);
```

```
else { fprintf(stderr,"teveel argumenten; ik kap ermee:
(\n"); exit(1); }

/** extraatje door martijn: **/
if(argc==3) heavystars=1;
else heavystars=0;

printf("SNU\n%d\n",nstar);

for (i=0 ; i<nstar ; i++)
{
         if(heavystars) /** [martijn] **/
         {
            if(i<10) M = 20./(nstar+190) ; // dit is het
            verschil met incon.c :
            eerste 10 sterren 10 maal zo zwaar als rest
            else M= 1./(nstar+190);        //
        }
        else M= 1./nstar;
        X1 = Random() ;
    r = 1./(sqrt(pow(X1,(-2./3.))-1.)) ;
    X2 = Random() ;
        z = (1.-2.*X2)*r ;
        X3 = Random() ;
        y = sqrt(pow(r,2.)-pow(z,2.))*sin(2*pi*X3) ;
        x = sqrt(pow(r,2.)-pow(z,2.))*cos(2*pi*X3) ;

    Ve = pow(sqrt(2.)*(1.+pow(r,2.)),(-1./4.)) ;

//printf("Ve=%f\n",Ve);

        X4 = Random() ;
```

```
            X5 = Random() ;

        while ( 0.1 * X5 >= g(X4) )
        {
            X4 = Random() ;
            X5 = Random() ;
        }

        q = X4 ;

    //printf("q= %f tienX5= %f g(x4)= %f\n",q,0.1*X5,g(X4));

        V = q*Ve ;
            X6 = Random();
        X7 = Random();
    //printf("x1= %f x2= %f x3= %f x4=%f x5=%f x6=%f x7=%f\n",X1,X2,X3,X4,X5,X6,X7);

            vz = (1.-2.*X6)*V ;
            vx = sqrt(pow(V,2.)-pow(vz,2.))*cos(2*pi*X7) ;
        vy = sqrt(pow(V,2.)-pow(vz,2.))*sin(2*pi*X7) ;

        printf("%lf %lf %lf %lf %lf %lf %lf\n",M,x,y,z,vx,vy,vz) ;

    }

    return 0;
}   double g (double q) {
    return (pow(q,2.)*pow((1.-pow(q,2)),(7./2.))) ;
}
```

B starsim.c + starsim.h

```
#include "starsim.h" /*****************************
***********/ /***** numerieke methoden: kies er 1  *****/
/*****************************************/
//#define evolve_cluster() ev_BOTTEBIJL()   /* v+=a*dt,x+=v*dt           */
//#define evolve_cluster() ev_LEAPFROG()    /* standaard leapfrog        */
#define evolve_cluster() ev_LEAPFROG_VT() /* tijdsymmetrische leapfrog 2e orde */
//#define evolve_cluster() ev_LEAPFROG4_VT() /* idem, 4e orde */

double dt; double currtime=0.; int nstar; int nshows; int evolutionspershow; int do_virialize; const
SI_G=6.6726e-11;

enum {
    SI = 1,
    SNU = 2
} unit=SI;

double sx1[nmax],sx2[nmax],sx3[nmax]; double sv1[nmax],sv2[nmax],sv3[nmax]; double sm[nmax];

void cluster_setup(void) /* laadt begincondities uit standaard input;
*/ /* maakt gemiddelde stersnelheid 0;
*/

/* FIXME: maak het punt met de grootste zwaartekrachtspotentiaal tot
*/ /* middelpunt, ipv het zwaartepunt
*/ {
    int i;
    double v1=0.,v2=0.,v3=0.;
    /* gemiddelde snelheid vd hoop; trek eraf bij iedere ster */
    double z1=0.,z2=0.,z3=0.;
```

```
/* massa-zwaartepunt vd hoop; trek van ieder ster-pos. af */
double m=0.;
/* totale massa bijhouden */
char units[20];

/* --- bestandsformaat: --- *
(eenheden)        (----- SI of SNU -----)
(aantal sterren)
(massa) (x-pos) (y-pos) (z-pos) (x-vel.) (y-vel.) (z_vel.)
(massa) (x-pos) (y-pos) (z-pos) (x-vel.) (y-vel.) (z_vel.)
    .
.
.
<etc>
*/
scanf("%s",units);
if(!strcmp(units,"SI"))
{unit =  SI; fprintf(stderr,"eenheden: SI\n"); }
else if(!strcmp(units,"SNU"))
{unit = SNU; fprintf(stderr,"eenheden: SNU\n"); }
else { fprintf(stderr,"eenheden niet bekend! We kappen ermee:(\n"); exit(1);}

scanf("%d",&nstar);
if(nstar>nmax)
{
    fprintf(stderr,"maximaal %d sterren!\n",nmax);
    exit(1);
}
fprintf(stderr,"aantal sterren %d\n",nstar);

for(i=0;i<nstar;i++)
    scanf("%lf %lf %lf %lf %lf %lf %lf",&sm[i],&sx1[i],&sx2[i],&sx3[i],&sv1[i],&sv2[i],&sv3[i]);
```

```
    for(i=0;i<nstar;i++)
    {
        m  += sm[i];
        v1 += sv1[i];
        v2 += sv2[i];
        v3 += sv3[i];
        z1  += sm[i] * sx1[i]; /* straks delen door totale massa */
        z2  += sm[i] * sx2[i];
        z3  += sm[i] * sx3[i];
    }
    v1 /= (double) nstar;
    v2 /= (double) nstar;
    v3 /= (double) nstar;
    z1 /= m;
    z2 /= m;
    z3 /= m;

    for(i=0;i<nstar;i++)
    {
        sv1[i] -= v1;
        sv2[i] -= v2;
        sv3[i] -= v3;
        sx1[i] -= z1;
        sx2[i] -= z2;
        sx3[i] -= z3;
    }

    if(do_virialize) virialize();
    if(unit==SI) SI_TO_SNU();
}
```

```
void show_cluster(void) /* schrijf cluster naar standaard out */ {
    int i;
    static int first_time=1;

    if(first_time) { first_time=0; printf("%d\n",nshows+1); }

    printf("%d\n",nstar);
    printf("%lf\n",currtime);
    for(i=0;i<nstar;i++)
        printf("%lf %lf %lf %lf %lf %lf %lf\n",sm[i],sx1[i],sx2[i],sx3[i],sv1[i],sv2[i],sv3[i]);

    fflush(stdout); /* als ie halverwege is, en cracht,
    is de data tenminste nog te gebruiken, evt */
}

void virialize(void) /* past snelheden vd sterren aan,
zodanig dat het cluster aan viriaalevenwicht voldoet */ {
    int i,j;
    double c;

    double T=0.,U=0.;

    /* bereken T */
    for(i=0;i<nstar;i++) T += .5*sm[i]*(pow(sv1[i],2.)+pow(sv2[i],2.)+pow(sv3[i],2.));

    /* bereken U */
    for(i=0;i<nstar;i++)
        for(j=i+1;j<nstar;j++)
        {
            double onedivr = pow( (sqr(sx1[i]-sx1[j])+sqr(sx2[i]-sx2[j])+sqr(sx3[i]-sx3[j])) ,-.5);
            U -= onedivr*sm[i]*sm[j]; /* interactie tussen sterren i en j levert -G m[i] m[j]/r(i,j)
        }
```

```
if(unit == SI) U*=SI_G;
fprintf(stderr,"voor virialize() T = %lf    U = %lf    T/U = %lf    E = %lf\n",T,U,T/U,T+U);

/* maak verhouding tussen T en U in orde */
c=-.5*U/T;      /* correctiefactor voor de posities */
for(i=0;i<nstar;i++)  /* maak viriaal-evenwicht */
{
    sx1[i]*=c;
    sx2[i]*=c;
    sx3[i]*=c;
}
U/=c;
//* hierna is E = T+U = 1/2 U   */

/* als SNU eenheden, zorg dan dat weer E = -1/4 ( correctiefactor voor T en U is -.25/E = -.5/U
if(unit==SNU)
{
    c=sqrt(-.5/U); /* correctiefactor voor snelheid/kin.energie */
    for(i=0;i<nstar;i++)
    {
        sv1[i]*=c;
        sv2[i]*=c;
        sv3[i]*=c;
    }
    c=-2.*U;       /* correctiefactor voor positie/pot.energie */
    for(i=0;i<nstar;i++)
    {
        sx1[i]*=c;
        sx2[i]*=c;
        sx3[i]*=c;
    }
}
```

```
    /********************** evt nog even testen ***************************************/

    T=0.; U=0.;
    for(i=0;i<nstar;i++) T += .5*sm[i]*(sqr(sv1[i])+sqr(sv2[i])+sqr(sv3[i]));
    for(i=0;i<nstar;i++)
        for(j=i+1;j<nstar;j++)
        {
            double onedivr = pow( (sqr(sx1[i]-sx1[j])+sqr(sx2[i]-sx2[j])+sqr(sx3[i]-sx3[j])) ,-.5);
            U -= onedivr*sm[i]*sm[j];
        }
    if(unit == SI) U*=SI_G;
    fprintf(stderr,"na virialize() T = %lf    U = %lf    T/U = %lf    E = %lf\n",T,U,T/U,T+U);


}

void SI_TO_SNU(void) {
    double SI_Ek,SI_Ep;
    double M=0.;  /* gaat de totale massa van het cluster bevatten (eenheid van energie in SI) */
    double E,L,T,V;
    int i,j;

    if(unit!=SI)
    {
        fprintf(stderr,"Eenheid is niet SI; wordt niet omgezet naar SNU\n");
        return;
    }


    for(i=0;i<nstar;i++) M += sm[i];

    SI_Ek=0.;
```

```
    for(i=0;i<nstar;i++) SI_Ek += .5*sm[i]*(sqr(sv1[i])+sqr(sv2[i])+sqr(sv3[i]));

    SI_Ep=0.;
    for(i=0;i<nstar;i++)
        for(j=i+1;j<nstar;j++)
        {
            double Gdivr= SI_G*pow( (sqr(sx1[i]-sx1[j])+sqr(sx2[i]-sx2[j])+sqr(sx3[i]-sx3[j])) ,-.5)
            SI_Ep -= Gdivr*sm[i]*sm[j]; /* interactie tussen sterren i en j levert -G m[i] m[j]/r(i,
        }

    E = SI_Ek+SI_Ep;                    /* totale energie */
    L = -SI_G*sqr(M)/(4.*E);        /* eenheid van lengte in SI eenheden */
    T = SI_G*pow(M,2.5)/pow(-4.*E,1.5); /* eenheid van tijd in SI eenheden */
    V = L/T;

    for(i=0;i<nstar;i++)
    {
        sm[i]  /= M;
        sx1[i] /= L;
        sx2[i] /= L;
        sx3[i] /= L;
        sv1[i] /= V;
        sv2[i] /= V;
        sv3[i] /= V;
    }

    unit=SNU;
}

void get_commandline(int argc, char *argv[]) {
    int i;
```

```
dt=1e-4;
evolutionspershow=100;
do_virialize=0;
nshows=100;

for(i=1;i<argc;i++)
{
    if(!strcmp(argv
    [i],"-v")) do_virialize=1;
    else if(!strcmp(argv[i],"-t")) sscanf(argv[++i],"%lf",&dt);
    else if(!strcmp(argv[i],"-n")) sscanf(argv[++i],"%d",&nshows);
    else if(!strcmp(argv[i],"-i"))
       {
           sscanf(argv[++i],"%d",&evolutionspershow);
           if(evolutionspershow<1)
           {
               fprintf(stderr,"i moet minimaal 1 zijn!\n");
               exit(1);
           }
       }
       else
    {
           fprintf(stderr,"opties: \n"
                          "-h  Help\n"
                          "-v  Breng de input tot viriaal-evenwicht\n"
                          "-t  Zet de tijdstap\n"
                          "-n  Geef op hoeveel data-outputs plaats moeten vinden\n"
                          "-i  Geef op hoeveel tijdstappen per data-output moeten worden doorger
           if(!strcmp(argv[i],"-h")) exit(0);
           else exit(1);
       }
  }
```

```
      fprintf(stderr,"%d data-outputs\nEr worden %d tijdstappen doorgerekend\n"
             "De tijdstap is %lf\n",nshows,nshows*evolutionspershow,dt);
}

void time_info(void) {
    static int k=0;
    double time_to;
    static clock_t old_clock=0,curr_clock; /* moet een beetje kloten, omdat clock_t snel tot */
    static double total_clock=0.;        /* overflow neigt -> negatieve tijd wachten. dus: gebruik d
    int jaar, maand, week, dag, uur, minuut, seconde;
    int total=evolutionspershow*nshows;

    k++;
    curr_clock=clock();
    total_clock += (double)(curr_clock-old_clock);
    old_clock=curr_clock;
    time_to=total_clock*(double)(total-k)/((double)CLOCKS_PER_SEC*(double)k);

    jaar=floor(time_to/31536000.);
    time_to-=jaar*31536000.;
    maand=floor(time_to/2592000.);
    time_to-=2592000.*maand;
    week=floor(time_to/604800.);
    time_to-=604800.*week;
    dag=floor(time_to/86400.);
    time_to-=86400.*dag;
    uur=floor(time_to/3600.);
    time_to-=3600.*uur;
    minuut=floor(time_to/60.);
    time_to-=60.*minuut;
    seconde=floor(time_to);
```

```
    fprintf(stderr,"\rvoortgang: %3.1f %% , nog %4d jr   %2d mnd   %2d wk   %2d d   %2d u   %2d m
    fflush(stderr);
}

int main(int argc,char *argv[]) {
    int i,j;

    get_commandline(argc,argv);
    cluster_setup();
    show_cluster();     /* eerste output is input */

    for(i=0;i<nshows;i++)
    {
        for(j=0;j<evolutionspershow;j++)
        {
            evolve_cluster();
            time_info();
        }
        show_cluster();
    }
    fprintf(stderr,"\n");

    return 0;
}
-----------------------------------------
#ifndef _starsim_h #define _starsim_h #include <stdio.h> #include
<stdlib.h> #include <string.h> #include <math.h> #include <time.h>
#include "integratie.h"

#define nmax 1024 /* allicht levert een macht van 2 van array-groottes snellere code op */

#define sqr(x) ((x)*(x)) /* eenvoudig aan te passen, mocht een andere manier sneller zijn */
```

```
void cluster_setup(void); void calca(void); void show_cluster(void); void virialize(void); void SI_T
void get_commandline(int argc, char *argv[]); void time_info(void);

#endif
```

C mathout.c

```
#include <stdio.h>

int main(void) {
    int i;
    int nshow;
    double xmin=0.,xmax=0.,ymin=0.,ymax=0.,zmin=0.,zmax=0.;


    FILE *XY=fopen("XY.nb","w+");
    FILE *YZ=fopen("YZ.nb","w+");

    fprintf(XY,"Table[ListPlot[{");
    fprintf(YZ,"Table[ListPlot[{");

    scanf("%d",&nshow); /* aantal data-eenheden */
    for(i=0;i<nshow;i++)
    {
        int j,n;
        double m,x1,x2,x3,v1,v2,v3;
        double currtime;

        fprintf(XY,"{");
        fprintf(YZ,"{");
```

```
        scanf("%d",&n);

        scanf("%lf",&currtime);
        for(j=0;j<n-1;j++)
        {
            scanf("%lf %lf %lf %lf %lf %lf %lf",&m,&x1,&x2,&x3,&v1,&v2,&v3);
            if(xmax<x1)xmax=x1; if(ymax<x2)ymax=x2; if(zmax<x3)zmax=x3;
            if(xmin>x1)xmin=x1; if(ymin>x2)ymin=x2; if(zmin>x3)zmin=x3;
            fprintf(XY,"{%lf,%lf},",x1,x2);
            fprintf(YZ,"{%lf,%lf},",x2,x3);
        }
        scanf("%lf %lf %lf %lf %lf %lf %lf",&m,&x1,&x2,&x3,&v1,&v2,&v3);
        if(xmax<x1)xmax=x1; if(ymax<x2)ymax=x2; if(zmax<x3)zmax=x3;
        if(xmin>x1)xmin=x1; if(ymin>x2)ymin=x2; if(zmin>x3)zmin=x3;
        fprintf(XY,"{%lf,%lf}},",x1,x2);
        fprintf(YZ,"{%lf,%lf}},",x2,x3);
    }

    fprintf(XY,"{{}} }[[i]],PlotRange->{{%lf,%lf},{%f,%f}},AspectRatio->1],{i,1,%d}];",xmin,xmax,ymi
    fprintf(YZ,"{{}} }[[i]],PlotRange->{{%lf,%lf},{%f,%f}},AspectRatio->1],{i,1,%d}];",xmin,xmax,ymi

    fclose(YZ);
    fclose(XY);

    return 0;
}
```

D gnuview.c

```
#include <stdio.h> #include "gnuplot_i.h"
```

```
#include <time.h>

void Sleep(int wait) {
    clock_t goal=wait*(CLOCKS_PER_SEC/1000)+clock();
    for(;goal>=clock(););
}

int main(int argc, char *argv[]) {
    int interleave=1;
    int i;
    int nshow;
    double currtime;
    FILE *tmp=tmpfile();
    gnuplot_ctrl *h1=gnuplot_init(),*h2=gnuplot_init();

    if(!tmp) fprintf(stderr,"GEEN TMPFILE. Nu ga ik crachen denk ik... :(\n");

    if(argc==2) sscanf(argv[1],"%d",&interleave);
    if(interleave<=0)
    {
        fprintf(stderr,"interleave moet positief zijn!\n");
        interleave=1;
    }

    scanf("%d",&nshow); /* aantal data-eenheden */
    for(i=0;i<nshow;i++)
    {
        int j,n;
        double t;
        int do_out=!(i%interleave);
        scanf("%d",&n);
```

```
    scanf("%lf",&t);

    if(do_out) fprintf(tmp,"%d\n%lf\n",n,t);

    for(j=0;j<n;j++)
    {
        double m,x1,x2,x3,v1,v2,v3;
        scanf("%lf %lf %lf %lf %lf %lf %lf",&m,&x1,&x2,&x3,&v1,&v2,&v3);
        if(do_out) fprintf(tmp,"%lf %lf %lf\n",x1,x2,x3);
    }
}
rewind(tmp);

gnuplot_cmd(h1,"set xrange [-12:12];set yrange [-12:12]");
gnuplot_cmd(h2,"set xrange [-12:12];set yrange [-12:12]");


for(i=0;i<nshow;i+=interleave)
{
    int j,n;
    FILE *gplotdata;

    gplotdata=fopen("gplot.dat","w+");

    fscanf(tmp,"%d",&n);
    fscanf(tmp,"%lf",&currtime);
    for(j=0;j<n;j++)
    {
        double x1,x2,x3;
        fscanf(tmp,"%lf %lf %lf",&x1,&x2,&x3);
        fprintf(gplotdata,"%lf %lf %lf\n",x1,x2,x3);
    }
```

```
        Sleep(100);
        gnuplot_cmd(h1,"plot \"gplot.dat\" using 1:2 title 'XY-aanzicht'");
        gnuplot_cmd(h2,"plot \"gplot.dat\" using 2:3 title 'YZ-aanzicht'");

        fclose(gplotdata);
    }

    fclose(tmp);
    gnuplot_close(h1);
    gnuplot_close(h2);
    remove("gplot.dat");
    return 0;
}
```

E clscat.c

```
/* waar zou dit nou weer voor zijn??

cl(u)s(ter)(con)cat(enation)

clscat [bestand 1] [bestand2]

clscat pakt twee bestanden met daarin ster-animaties, en voegt
deze samen tot 1 met behulp van clscat en inconfromcls kan je na
een simulatie besluiten de simulatie uit te breiden tot verdere
tijd

ER WORDT VANUIT GEGAAN DAT DE LAATSTE HOOP VAN BESTAND1 GELIJK IS
AAN DE EERSTE VAN DIE VAN BESTAND 2; DAAROM WORDT DE EERSTE VAN 2
OVERGESLAGEN
```

```
ER WORDT VANUIT GEGAAN DAT ZOWEL BESTAND 1 ALS BESTAND 2 OP t=0
BEGINNEN */

#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *in1,*in2;
    int n1,n2;
    int nstar;
    double time,addtime;
    int i,j;

    if(argc!=3) return 1;

    in1=fopen(argv[1],"r");
    in2=fopen(argv[2],"r");

    if(!in1 || !in2) return 2;

    fscanf(in1,"%d",&n1);
    fscanf(in2,"%d",&n2);

    printf("%d\n",n1+n2-1);

    for(i=0;i<n1;i++)
    {
        double m,x1,x2,x3,v1,v2,v3;

        fscanf(in1,"%d %lf",&nstar,&time);
        printf("%d\n%lf\n",nstar,time);
```

```
        for(j=0;j<nstar;j++)
        {
            fscanf(in1,"%lf %lf %lf %lf %lf %lf %lf",&m,&x1,&x2,&x3,&v1,&v2,&v3);
            printf("%lf %lf %lf %lf %lf %lf %lf\n",m,x1,x2,x3,v1,v2,v3);
        }
    }

    addtime=time;

    for(i=0;i<n2;i++)
    {
        double m,x1,x2,x3,v1,v2,v3;

        fscanf(in2,"%d %lf",&nstar,&time);

        if(i!=0) printf("%d\n%lf\n",nstar,time + addtime);

        for(j=0;j<nstar;j++)
        {
            fscanf(in2,"%lf %lf %lf %lf %lf %lf %lf",&m,&x1,&x2,&x3,&v1,&v2,&v3);
            if(i!=0)printf("%lf %lf %lf %lf %lf %lf %lf\n",m,x1,x2,x3,v1,v2,v3);
        }
    }

    fclose(in2);
    fclose(in1);

    return 0;
}
```

F inconfromcls.c

```
/* programma geeft begincondities op basis van de laatste
sterrenhoop in een output van starsim */

#include <stdio.h>

int main(void) {
    int n,i,j,ns;
    double t;
    printf("SNU\n");

    scanf("%d",&n);

    for(i=0;i<n;i++)
    {

        scanf("%d %lf",&ns,&t);
        if(i==(n-1)) printf("%d\n",ns);

        for(j=0;j<ns;j++)
        {
            double m,x1,x2,x3,v1,v2,v3;
            scanf("%lf %lf %lf %lf %lf %lf %lf",&m,&x1,&x2,&x3,&v1,&v2,&v3);
            if(i==(n-1)) printf("%lf %lf %lf %lf %lf %lf %lf\n",m,x1,x2,x3,v1,v2,v3);
        }
    }

    return 0;
}
```

G getinit.c

```
#include <stdio.h>

int main(void) {
    int idummy,nstar,i;
    double fdummy;
    scanf("%d %d %lf",&idummy,&nstar,&fdummy);
    printf("%d\n",nstar);
    for(i=0;i<nstar;i++)
    {
        double m,x1,x2,x3,v1,v2,v3;
        scanf("%lf %lf %lf %lf %lf %lf %lf",&m,&x1,&x2,&x3,&v1,&v2,&v3);
        printf("%lf %lf %lf %lf %lf %lf %lf\n",m,x1,x2,x3,v1,v2,v3);
    }

    return 0;
}
```

H trapezium.c

```
#include <stdio.h> #include <math.h>

FILE *tdf;

int main(int argc, char *argv[]) {
  double a, b=0.;
  double n = 1000;
```

```
double rc=1.;
double c;
double G =1.;
double M=1.;
double chi=0.3406;
double lnlabda= log(0.1*n);
double pi = 4*atan(1.);
double h;
double xi;
double m;
double valint;
int nstep = 100;
int i,j;
double r = a;
double dr;
char title[] = "Grafiek";

printf("Geef de afstand van de te volgen ster:\n");
scanf("%lf", &a);
dr = a/nstep;
printf("Geef de massa van de inspiralizerende ster:\n");
scanf("%lf", &m);
  //  printf("a= %lf \n",a);
c=(sqrt(M/G))*(2.*pi)/(chi*lnlabda*m*pow(rc,7./2.));
//  printf("c= %lf \n", c);

b = a;
tdf=fopen("tdf.nb","w+");
fprintf(tdf, "ListPlot[{");
j=0;
do
{
```

```
    b = b - dr;

    h=(b-a)/nstep;
    //  printf("h=: %lf %lf %lf %lf\n",h, b, a, dr);
    valint = c*(pow(a,4.))/(1+pow(a,2.)/pow(rc,2.)) + c*(pow(b,4.))/(1+pow(b,2.)/pow(rc,2.));
    //    printf("tussen: %lf \n",valint);

    for(i=1; i<nstep; i=i+1)
    {
        xi = a + i*h;
        valint = valint + 2. *c*(pow(xi,4.))/(1+pow(xi,4.)/pow(rc,4.));
    //  printf("tussen: %lf \n",valint*0.5*h);
    }

    valint = -h/2. * valint;
    printf(" t= %lf r= %lf \n",valint,b);
    fprintf(tdf, "{%f,%f}", valint,b);
    if(j<(nstep-1))
      {
          fprintf(tdf,",");
      }
     j=j+1;
  }
  while(b > 0);

  fprintf(tdf, "},PlotJoined->True]");
  fclose(tdf);
  return 0;
}
```

I Montgomery + Pythagoras init.cond.

Montgomery:

```
SNU 3 1 -.97000436000000001 .243087529999999996 0
-.466203684999999979 -.432365730000000004 0 1 .97000436000000001
-.243087529999999996 0 -.466203684999999979 -.432365730000000004 0
1 0 0 0 .932407369999999958 .864731460000000007 0
```

Pythagoras:

```
SNU 3 3 1 3 0 0 0 0 4 -2 -1 0 0 0 0 5 1 -1 0 0 0 0
```

J Figure 3 from Bonnell and Davies

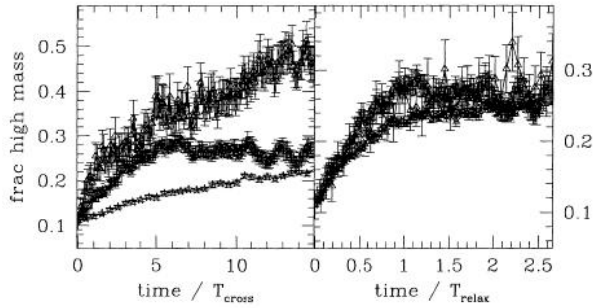

Figure 3. The fraction of stars in the inner half of a half-mass radius having a high mass ($M > 1$ M$_\odot$) versus time for (from top to bottom) clusters with 50, 250 and 1000 stars. The right-hand panel shows the same curves, where now time is given in units of the relaxation time for each cluster.

Figure 7: Taken from Bonnell and Davies, 1998 [1]. They use crossing times (t_{cr}). One t_{cr} corresponds with $2\sqrt{2}$ SNU units.