Particle Simulation II

Course Assignment

Andrew N Jackson

14th Feburary 1997

(1) Introduction

The aims of this assignment are two-fold. Firstly, to write a two-dimensional collisionless particle-in-cell code which simulates the behaviour of N identical masses interacting via the gravitational force, and secondly, to use this program to simulate some aspect of the evolution of galaxies. The problem of galactic evolution can be simulated in this way via the following model.

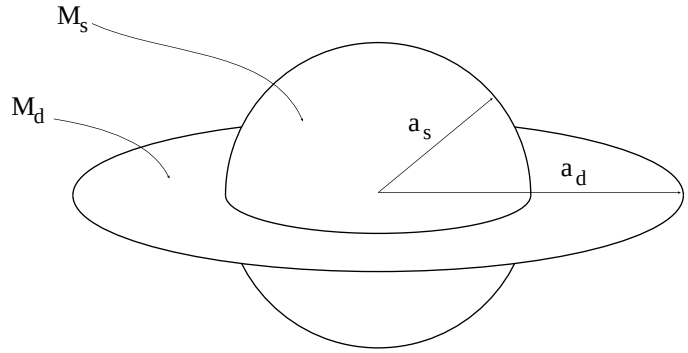


Figure 1: Simplified model of a galaxy. M_s, a_s, M_d and a_d are the masses and radii of the spherical halo and the galactic disc respectively.

The general structure of a galaxy is broken down into two parts (see figure 1), the spherically symmetric halo (with radius a_s and mass M_s), and the galactic disc (radius a_d, mass M_d). The character of any particular galaxy can then be defined by values of ρ and α, where:

$$\rho = M_d / (M_s + M_d)$$

and

$$\alpha = a_d / a_s$$

This simplification allows a 2-D system to be used to simulate the galactic disc, with the force due to the spherically symmetric halo superimposed on top of it. Also, if we use a collisionless model, we can use stars of identical mass (the average stellar mass), as opposed to the distribution of masses of stars in a real galaxy.

Based of the above model, the specific aims of this assignment are to write a 2-D collisionless particle-in-cell code for the galactic disc, and to use this to prove that a disc with no halo (ie when $\rho=0$) is unstable, and that given a suitable halo (eg $\rho=0.2$, $\alpha=1.0$), the disc is not only stable, but also capable of forming filamentous or spiral structures.

(2) Theory

Particle Dynamics:

From the model the problem reduces to a disc of equally massive particles, evolving under Newton's law of motion and interacting via Newton's law of gravity with the force due to the halo superimposed upon the particle-particle attraction. For each particle pair:

$$\underline{F} = \frac{Gm^2}{r^2}.\hat{\underline{r}} \qquad -(1)$$

With every particle moving such that:

$$\frac{d^2\underline{r}}{dt^2} = \frac{\underline{F}}{m} \qquad -(2)$$

where F_t is the total overall force on that particle.

Initial Conditions:

Because of the way in which clouds of galactic matter come together, the best initial condition for the disc of stars of not a uniform distribution, but a radially varying rotationally symmetric distribution of the form:

$$n(r) = n(0)\sqrt{1-\frac{r^2}{a_s^2}} \qquad -(3)$$

This mass distribution applies both to the disc and the halo. In order to determine the force due to the halo at a point inside it, we must integrate equation (3) to find the mass contained within a sphere inside the halo as a function of the radius of that sphere. This gives a mass distribution of the form:

$$M_s(r) = M_s(1-\cos^3\theta) \qquad -(4)$$

where:

$$\sin\theta = \frac{r}{a_s} \qquad -(5)$$

For a particle at a radial distance r, the force it feels inside the halo is the same as that from a point mass equivalent in magnitude to the mass enclosed by a sphere or radius r. This means that the force due to the halo is:

$$\underline{F} = \frac{GmM_s(r)}{r^2}.\hat{\underline{r}} \qquad -(6)$$

for $r < a_s$, and:

$$\underline{F} = \frac{GmM_s}{r^2}.\hat{\underline{r}} \qquad -(7)$$

for $r \bullet a_s$.

While we now know the forces and initial positions, we need the initial velocities to be defined in order to calculate the disc's evolution through time. When considering the particle velocities in a model such as the one used here, the disc is usually defined as being either 'cold' or 'hot':

- **Cold disc:** Each star has no radial velocity and just enough angular velocity to counterbalance the gravitational force on it.
- **Hot disc:** As for the cold disc, except the radial velocity is not zero, but a random set of velocities corresponding to a thermal (gaussian) distribution.

While the hot or cold nature of a disc can have important consequences for the disc's stability (hot discs with no halo can be more stable than cold discs without halos), only the evolution of cold discs will be investigated in this assignment.

(3) Method

While a full 3-D simulation would involve finding the gravitational potential over a 3-D mesh encompassing the system, this approach does not work in 2-D. In this case, the particle methods solution works as follows: given the particle distribution, the mass of the particles is distributed over a 2-D mesh. Using this set of point masses we can find the total force on each mesh point by vector summation of the forces due to the masses at each other mesh point, the forces being calculated from equation 1..

To simulate a galaxy using the same number of particles as there are in an average galaxy ($\sim 10^{10}$) would make the program impractical, as it would take far too long to run. To get around this, the particle-in-cell code implements the standard technique of using 'superparticles', each of which corresponds to a large number of stars ($\sim 10^5$) moving together. This has no detrimental effect on the results provided that the number of superparticles is still quite large (in this case $\sim 10^5$). Another technique to make the program faster and simpler is the use of dimensionless coordinates, which avoids the presence of very large or very small numbers in the calculation, and removes the repeated calculation of unchanging quantities from the code. Given fundamental units of length Δ (in this case the horizontal and vertical separation of the mesh points) and of time $2\Delta t$, (the unit of time is $2\Delta t$ as opposed to Δt due to the nature of the integration of the equations of motion, see step 5 below). If we mark the dimensionless form of position, velocity and acceleration as x', v' and a', we convert from x, v, a to dimensionless form as follows:

$$x` = x/\Delta$$

$$v' = v.2\Delta t/\Delta$$

$$a' = a.4(\Delta t)^2/\Delta$$

Tranforming out system via this substitution, we find that the Δt in the force equations and velocity conditions is accompanied by ω_p, where:

$$w_p = SQRT(4\pi Gmn)$$

where n is the number density:

$$n = [\text{Total number of particles}] / [\text{Total area of the mesh}\}$$

The Δt and ω_p parameters are then replaced by one, such that:

$$DT = \omega_p \Delta t$$

Where DT is the dimensionless time-step.

Initial Conditions

The dimensionless initial positions of the atoms are set up as follows. For a mesh of resolution MSxMS, the valid spatial range for x and y is 0-MS, and so the centre of the mesh is at (MS/2,MS/2). A rotationally symmetric distribution of stars (as defined by equation 3) is placed around the centre-point, such that the diameter of the disc is half the total width/hieght of the mesh (ie MS/2). This is achieved by breaking the total radius a_d into m steps of a_d/m. A numerical integration is then performed to find the value of n_o for this situation (ie number of particles and disc radius), and then particles are randomly distributed in each of the m radial sections, each with a random angle in the range 0 - 2π.

In order to find the initial velocities, we must carry out some calculations to find the forces on the particles. This is because in a cold disc each particle has just enough energy to counterbalance the centripetal gravitational acceleration. In other words, the magnitide of the velocity carries the condition that

$$a' = v'^2 / r'$$

or

$$v' = \bullet(a'r')$$

The acceleration on each particle can be found from steps 1-4 below, and then the speed each particle requires can be calculated from the above equation and then given to the particle as a tangential velocity.

By the end of the initialisation, we have set up the arrays:

$x(NP), y(NP), V_x(NP), V_y(NP)$

Where NP is the number of particles.

Step 1: Assignment of particle masses to mesh points.

In the dimensionless form, we do not need the mass distribution, but the number distribution of particles (see step 2). There are two main techniques for distributing the particles onto the mesh, Nearest Grid Point (NGP) and Cloud-In-Cell (C-in-C). In the NGP technique, each particle is added into the particle count for a single mesh point (the nearest one), while the C-in-C method splits the particle between the 4 nearest points, and distributes it according to how close to each surrounding mesh point it is. While C-in-C is more accurate, NGP has been used in this case.

In other words, this step translates particle positions into a mesh distribution:

$$x(n), y(n) \quad \Pi \quad n(i,j)$$

Where n is a matrix of the number distribution, of dimensions MSxMS.

Step 2: Calculation of the forces at each mesh point.

This calculation is based on finding the force between each pair of mesh points. The simplest way would be to use two nested loops over all mesh points, but this would be rather inefficent. If we consider the forces between point (i,j) and point (i',j'), then the force on (i,j) is equal and opposite to that on (i',j'), and there is no point is calculating the force twice. This means we instead use two nested loops where the first loop covers all the mesh points (i=0≈32,j=0≈32), and the second only covers the remaining points (i'=i+1≈32,j'=j≈32). We do not consider the case where i=i', j=j'.

In dimensionless form, the magnitude of the force between two mesh points is:

$$|a(i,j)| = n(i',j') \cdot \frac{DT^2}{4\pi N_o} \cdot \frac{1}{r'^2}$$

Where N_o is the average number of particles per cell, and r' is the dimensionless distance between cells (i,j) and (i',j'). For this problem, we are working in 2-D, and so the horizontal and vertical components of the inter-mesh-point force are:

$$a_x(i,j) = |a(i,j)| \cdot x'/r'$$

$$a_y(i,j) = |a(i,j)| \cdot y'/r'$$

Where x' and y' are the horizontal and vertical separation of the mesh points (i,j) and (i',j'). As we are referring to a set of mesh points and not free points in space, the number of different combinations of x', y' and r' is not infinite, but equal to the number of different combinations of (i,j) and (i',j'). This means that to further speed up the calculation a look-up table of all the relevant coefficients can be created.

Step 3: Assignment of accelerations to particles.

This step is exactly analogous to step 1, and must be carried out by the same method in order for the calculation to have meaningful results. In this case that mean that all particles whose nearest mesh point is at i,j must all have the same acceleration made up of $a_x(i,j)$ and $a_y(i,j)$. Doing this for all particles means that we have performed the back translation:

$$a_x(i,j),\ a_y(i,j) \quad \Pi \quad a_x(n),\ a_y(n)$$

Step 4: Adding in the acceleration due to the halo.

This step is similar to step 2 above, but in this case we step through all the particles and add on to it's acceleration the force due to the halo. The dimensionless form of this force is worked out as follows:

If we have NP stars in the disc, then the number of stars in the halo (NH) is worked out as:

$$NH = NP \ . \ (1 - \rho)/\rho$$

And it's radius is:

$$a_s = a_d/\alpha$$

Equation 4 becomes:

$$NH(r) = NH\ (1-\cos^3\theta)$$

Where θ is as defined in equation 5. This means that the magnitudes of the forces in equations 6 and 7 become:

$$|a| = NH(r) \ . \ \frac{DT^2}{4\pi N_o} \ . \ \frac{1}{r'^2}$$

for $r < a_s$, and:

$$|a| = NH \ . \ \frac{DT^2}{4\pi N_o} \ . \ \frac{1}{r'^2}$$

for $r \bullet a_s$, where r' is the distance between the current particle and the centre of the halo. The modifications to $a_x(n)$ and $a_y(n)$ can be easily worked out from this.

Step 5: Integration of the equations of motion of the particles:

The integration algorithm used here is the simple Leapfrog method, which has the following dimensionless form:

$$v_y(n)^{t-1} = v_y(n)^{t-3} + a_y(n)^{t-2}$$

$$v_x(n)^{t-1} = v_x(n)^{t-3} + a_x(n)^{t-2}$$

$$y(n)^{t} = y(n)^{t-2} + v_y(n)^{t-1}$$

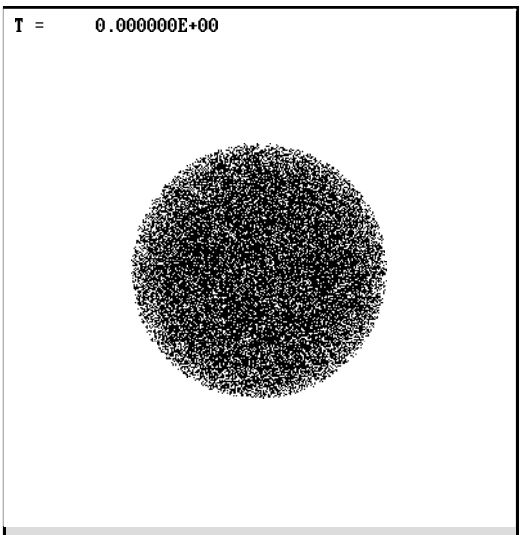
$$x(n)^{t} = x(n)^{t-2} + v_x(n)^{t-1}$$

Where t is the integer time count. Using this algorithm means that we need to know the positions at a different time to the velocities (t-2, compared to t-3), which can be taken account of by a simple Euler step.

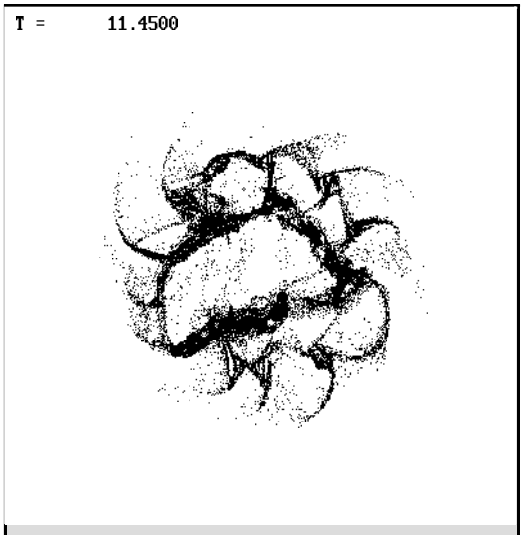
The value of DT is critical for the success of the simulation, and while the theory gave a guideline of DT ⌋ 0.1, the actual best value depends on the number of particles and the resolution of the mesh. In this assignment I use NP=25,000 particles and a mesh resolution of MS=32, for which I found DT=0.05 to be a suitable time-step.

(4a) Results for a Galactic Disc with no Halo:

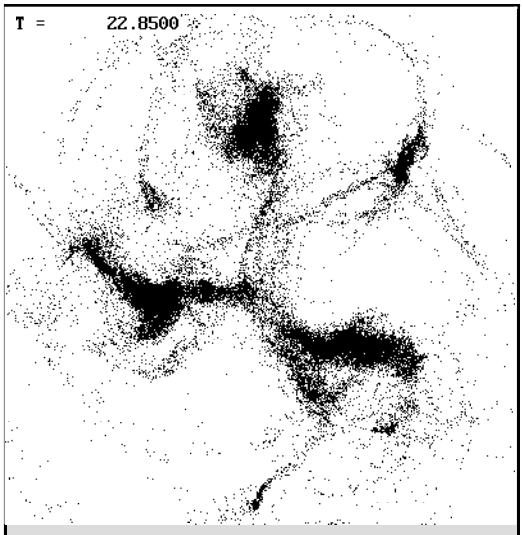
Plots 1.1 to 1.4 below show the the evolution of a galactic disc without a halo (ρ=0) for 3 periods of galactic rotation (this period being based on the assumption that the disc rotates as a ridged body). Simulated with NP= 25,000, MS=32 and Dt=0.05.



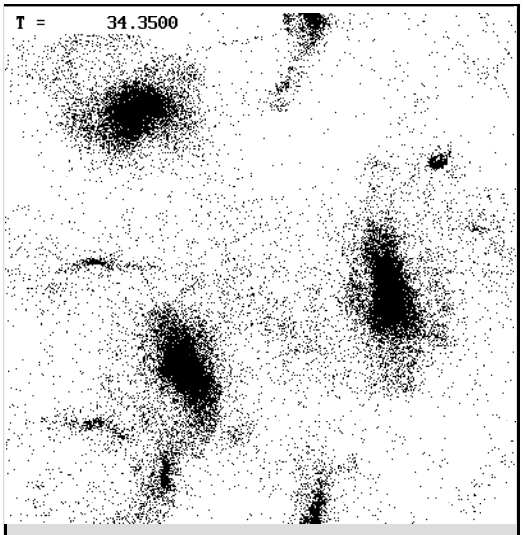
Plot 1.1: Initial distribution.



Plot 1.2: After 1 rotation period.



Plot 1.3: After 2 rotation periods.

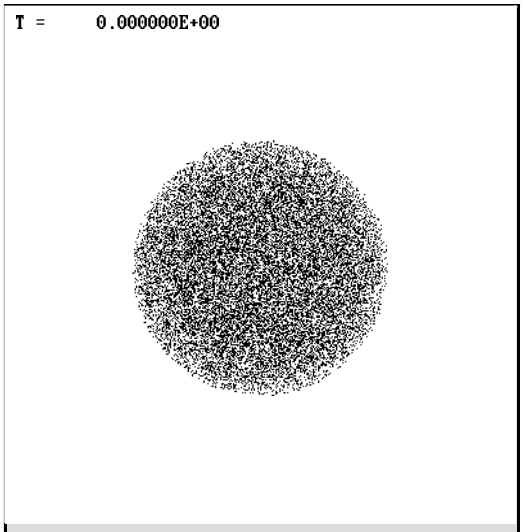


Plot 1.4: After 3 rotation periods.

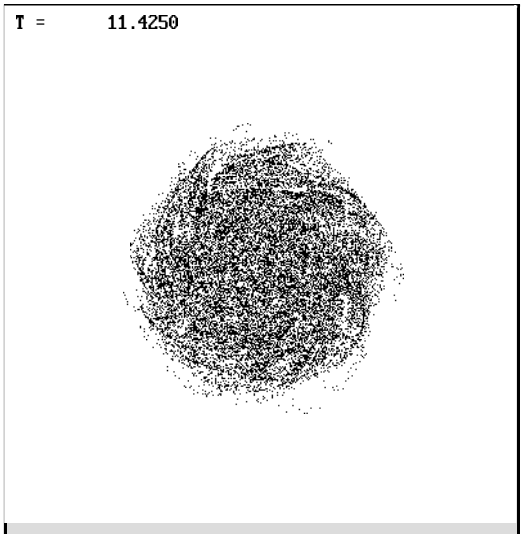
It is clear from these results that the disc of stars is completely unstable, in agreement with the work by others in this field (see references). This is known as the Jean's instability, and is cause by local maxima in the star distribution, which then draw more particles in and consequently lead to the break-up of the disc.

(4b) Results for a Galactic Disc Heavy Diffuse Halo:

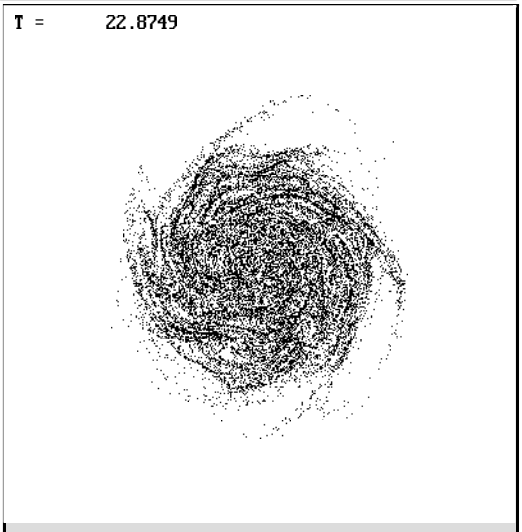
Plots 2.1 to 2.7 below show the the evolution of a galactic disc with a heavy, diffuse halo (ρ=0.2, α=1.0) for 6 periods of galactic rotation. NP=25,000, MS=32 and Dt=0.05 as before.



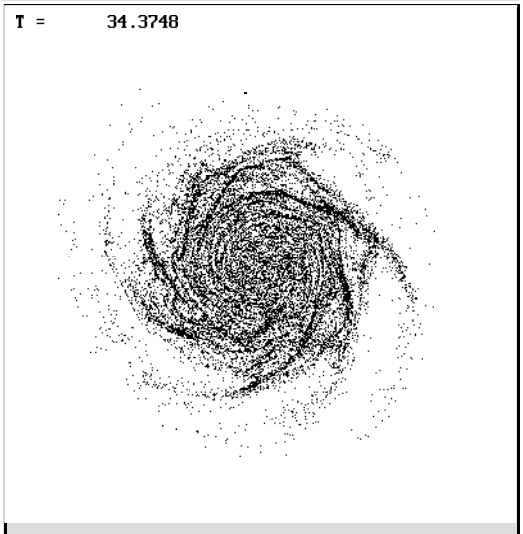
Plot 2.1: Initial distribution.



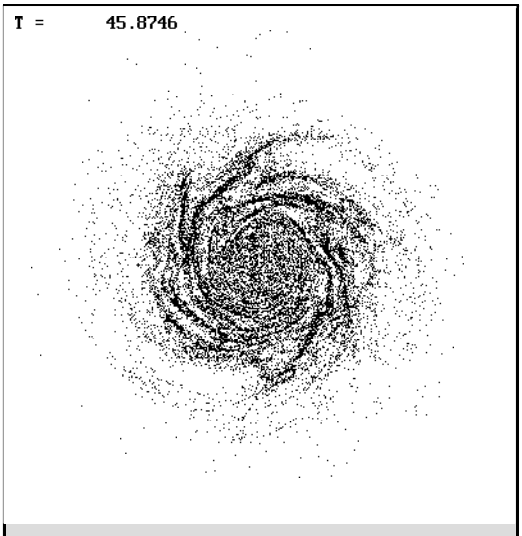
Plot 2.2: After 1 rotation.



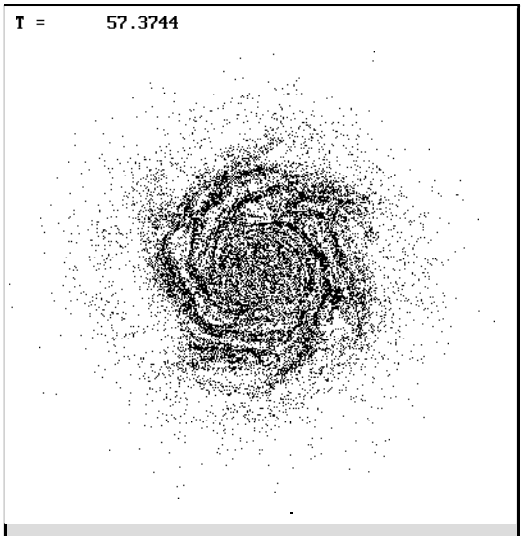
Plot 2.3: After 2 rotations.



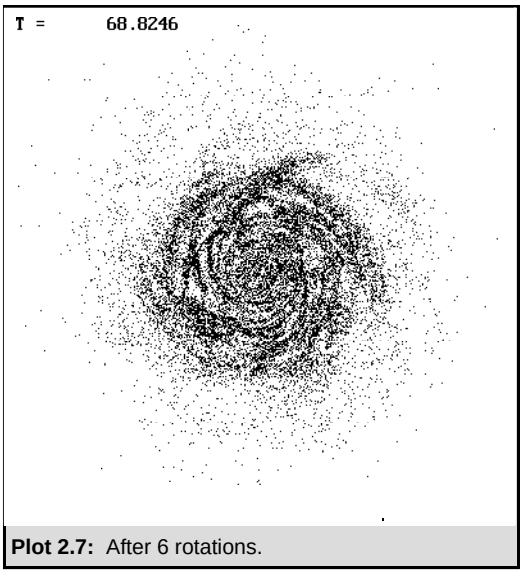
Plot 2.4: After 3 rotations.



Plot 2.5: After 4 rotations.



Plot 2.6: After 5 rotations.



Plot 2.7: After 6 rotations.

From these plots it is clear that the disc is not only stabilised by the halo, but it also allows the formation of filamentous structure making these results look very much as one might expect a spiral galaxy to look. This spiral structure is not actually formed by density waves, as in some galaxies, but by localised Jean's instability effects which the halo stabilises, thus forming strands and avoiding the formation of larger bunches of stars as in the case of the disc with no halo.

(5) Conclusion

In this assignment it has been successfully shown that using a 2-D N identical body particle-in-cell code is is possible to illustrate the stabilising effects of a halo on the evolution of a galactic disc. While this represents a success, any serious examination of galactic evolution via particle methods would have to use a 3-D simulation, as the 2-D form does not make a great approximation in some cases. The spiral structure in a 3-D simulation is longer lived and more physically realistic. Better results could be gained from the 2-D simulation by using more particles (NP~50,000), a finer mesh (MS~64) and a correspondingly lower time-step (DT~0.01), but the run-time on a desktop PC for such a program is large (~24hrs for 2 galactic rotations) and so it was considered impractical for this assignment.

Bibliography

Hockney R.W. & Eastwood J.W. (Adam Higler) Computer simulation using particles.

Hockney R.W. & Browrigg D.R.K. (1974) Effect of population II stars and three-dimensional motion on spiral structure (Mon. Not. R. astr. Soc. **167**, pp 351-357).

Berman R.H., Browrigg D.R.K. & Hockney R.W. (1978) Numerical models of galaxies - I. The vraibility of spiral structure (Mon. Not. R. astr. Soc. **185**, pp 861-875).

James R.A. & Sellwood J.A. (1978) Galactic models with variable spiral structure (Mon. Not. R. astr. Soc. **182**, pp 331-344).

Miller R.H. (1976) Predominance of two-armed spirals (The Astrophysical Journal, **207**, pp 408-413).

Appendix A: FORTRAN program code:

```
C      PROGRAM Galactasim
C      Andrew Jackson: v1.4 24th January 1997.
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      COMMON /GRAINT/ SCRNN
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,rh,NH
      INTEGER M
      INTEGER*2 K
      INTEGER*4 SCRNN
C
      REAL tbrk,tstep
      K=0
C
      CALL SysSetup
      CALL InitConds(tbrk,tstep)
      CALL PlotStars
      CALL Euler

  666   WRITE(*,*) 'Working...'

  999     CALL NGPStarsToMesh
          CALL ForceCalc
          CALL NGPForceFromMesh
          CALL HaloPot
          CALL LeapFrog
        IF (t.LT.tbrk) GOTO 999
        tbrk=t+tstep
        CALL PlotStars
c        CALL GET_KEY@(K)
      IF (K.NE.Z'13B' .AND. SCRNN.LE.20) GOTO 666
      CALL TEXT_MODE@
      STOP
      END
C
C ---------------------------------------------------------------------------
C
      SUBROUTINE SysSetup
      COMMON /GRAINT/ SCRNN
      INTEGER*4 SCRNN
C
C Set up graphics mode etc etc...
      SCRNN=1
      CALL VGA@
C
      END
C
C ---------------------------------------------------------------------------
C
      SUBROUTINE InitConds(tbrk,tstep)
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
```

```
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,rh,NH,omega,No
      REAL tbrk,tstep
      REAL Rad,Len,Ang,r,bit,Noo,RAN1,alpha,rho,tx,ty,tr,acc,vel
      INTEGER M,IDUM,S,SS,N,b,bits
C
      IDUM=-1
      t=0.0
      tstep=11.4*0.5
      tbrk=t+tstep
      dt=0.01
      No=REAL(NP)/REAL(MS*MS)
      Rad=1000.0
      Len=2.0*Rad
      Len=2.0*Len
      dx=Len/MS
      rho=0.2
      alpha=1.0
      rh=(Rad/alpha)/dx
      NH=NP*(1.0-rho)/rho
C
      S=1
C
      bits=NP/100
      bit=Rad/REAL(bits)
      Noo=0.0
C
      DO b=0,bits-1
       r=b*bit
       Noo=Noo+SQRT(1-(r*r/(Rad*Rad)))
      ENDDO
      Noo=NP/Noo
      omega=SQRT(REAL(Noo)/REAL(2*NP))
      omega=0.5513288
C
      DO b=0,bits-1
       r=b*bit
       N=INT(Noo*SQRT(1-(r*r/(Rad*Rad))))
       DO SS=1,N
        r=bit*(b+RAN1(IDUM))
        r=SQRT(Rad*r)
        Ang = 2.0*3.1415962*RAN1(IDUM)
        x(S)=MS/2 + r*COS(Ang)/dx
        y(S)=MS/2 + r*SIN(Ang)/dx
        S=S+1
       ENDDO
      ENDDO
C
      DO SS=S,NP
        Ang = 2.0*3.1415962*RAN1(IDUM)
        r   = Rad*RAN1(IDUM)
        r   = SQRT(Rad*r)
        x(SS)=MS/2.0 + r*COS(Ang)/dx
        y(SS)=MS/2.0 + r*SIN(Ang)/dx
      ENDDO
C
```

```
C Calc cold disc velocities from ax's:
      CALL NGPStarsToMesh
      CALL ForceCalc
      CALL NGPForceFromMesh
      CALL HaloPot
C
      DO S=1,NP
       tx=(x(S)-MS/2.0)*dx
       ty=(y(S)-MS/2.0)*dx
       tr=SQRT(tx*tx+ty*ty)
       acc=SQRT(ax(S)*ax(S)+ay(S)*ay(S))
       vel=SQRT(acc*tr/dx)
       vx(S)=vel*ty/tr
       vy(S)=-vel*tx/tr
      ENDDO
      END
C
C ---------------------------------------------------------------------
C
      SUBROUTINE HaloPot
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,tx,ty,tr,No,C,rh,NH
      REAL NewNH,Ang
      INTEGER M,S
C
      No=REAL(NP)/REAL(MS*MS)
      C=dt*dt/(4.0*3.141592654*No)

      DO S=1,NP
        tx=(MS/2)-x(S)
        ty=(MS/2)-y(S)
        tr=SQRT(tx*tx+ty*ty)
        IF (tr.LE.rh) THEN
         Ang=ASIN(tr/rh)
         NewNH=NH*(1.0-(COS(Ang)**3))
        ELSE
         NewNH=NH
        ENDIF
         ax(S)=ax(S)+C*NewNH*tx/(tr*tr*tr)
         ay(S)=ay(S)+C*NewNH*ty/(tr*tr*tr)
       ENDDO
      END
C
C ---------------------------------------------------------------------
C
      SUBROUTINE Euler
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,rh,NH
```

```
      INTEGER M,S
C
      DO S=1,NP
        x(S)=x(S) + 0.5*vx(S)
        y(S)=y(S) + 0.5*vy(S)
      ENDDO
      t=t+dt
      END
C
C ---------------------------------------------------------------------
C
      SUBROUTINE PlotStars
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      COMMON /GRAINT/ SCRNN
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,rh,NH
      INTEGER M,S
      INTEGER*2 IH,IV,IR,ICOL,ERR_C
      INTEGER*4 BUFFER,SCRNN
      CHARACTER*12 FILE,CNUM
C
      ICOL=15
      CALL VGA@
C
      DO S=1,NP
        IH=400*x(S)/MS
        IV=400*y(S)/MS
        CALL SET_PIXEL@(IH,IV,ICOL)
      ENDDO
C
      S=INT(NP/2)
      ICOL=10
      IR=3
      IH=400*x(S)/MS
      IV=400*y(S)/MS
c      CALL FILL_ELLIPSE@(IH,IV,IR,IR,ICOL)
C
      WRITE(*,*) 'T = ',t
      FILE(4:8) = CNUM(SCRNN+10000)
      FILE(1:4) = 'SCRN'
      FILE(9:12)= '.PCX'
      CALL GET_SCREEN_BLOCK@(0,0,639,479,BUFFER)
      CALL SCREEN_BLOCK_TO_PCX@(FILE,BUFFER,ERR_C)
      SCRNN=SCRNN+1
C
      END
C
C ---------------------------------------------------------------------
C
      SUBROUTINE NGPStarsToMesh
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
```

```
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,rh,NH
      INTEGER M,I,J,S
C
      DO J=1,MS
        DO I=1,MS
          M(I,J)=0.0
        ENDDO
      ENDDO
C
      DO S=1,NP
        I=INT(x(S))+1
        J=INT(y(S))+1
        M(I,J)=M(I,J)+1
      ENDDO
      END
C
C ---------------------------------------------------------------------
C
      SUBROUTINE  ForceCalc
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,rh,NH,dex,dey,der,C
      INTEGER M,I,J,II,JJ
C
      DO J=1,MS
        DO I=1,MS
          MSax(I,J)=0.0
          MSay(I,J)=0.0
        ENDDO
      ENDDO
C
      No=REAL(NP)/REAL(MS*MS)
      C=dt*dt/(4.0*3.141592654*No)
      DO J=1,MS
       DO I=1,MS
        DO JJ=J,MS
         DO II=1,MS
          IF ((JJ.EQ.J .AND. II.GT.I) .OR. JJ.GT.J) THEN
          IF (II.NE.I .OR. JJ.NE.J) THEN
           dex=REAL(II-I)
           dey=REAL(JJ-J)
           der=SQRT(dex*dex+dey*dey)
           MSax(I,J)=MSax(I,J)+M(II,JJ)*C*dex/(der*der*der)
           MSay(I,J)=MSay(I,J)+M(II,JJ)*C*dey/(der*der*der)
           MSax(II,JJ)=MSax(II,JJ)-M(I,J)*C*dex/(der*der*der)
           MSay(II,JJ)=MSay(II,JJ)-M(I,J)*C*dey/(der*der*der)
          ENDIF
          ENDIF
         ENDDO
        ENDDO
       ENDDO
      ENDDO
```

```
      END
C
C ---------------------------------------------------------------------
C
      SUBROUTINE NGPForceFromMesh
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,rh,NH
      INTEGER M,S
C
      DO S=1,NP
        I=INT(x(S))+1
        J=INT(y(S))+1
        Ax(S)=MSax(I,J)
        Ay(S)=MSay(I,J)
      ENDDO
      END
C
C ---------------------------------------------------------------------
C
      SUBROUTINE LeapFrog
C
C Define common variables:
      PARAMETER (NP=50000,MS=64)
      DIMENSION x(NP),y(NP),vx(NP),vy(NP),ax(NP),ay(NP)
      DIMENSION M(MS,MS),MSax(MS,MS),MSay(MS,MS)
      COMMON /GALREL/ x,y,vx,vy,ax,ay,M,MSax,MSay,dt,dx,t,rh,NH
      REAL x,y,vx,vy,ax,ay,MSax,MSay,dt,dx,t,rh,NH
      INTEGER M,S
C
      DO S=1,NP
        vx(S)=vx(S) + ax(S)
        vy(S)=vy(S) + ay(S)
        x(S)=x(S) + vx(S)
        y(S)=y(S) + vy(S)
C No leavy meshmesh...
        IF (x(S).GT.MS) x(S)=x(S)-MS
        IF (x(S).LT.0.0) x(S)=x(S)+MS
        IF (y(S).GT.MS) y(S)=y(S)-MS
        IF (y(S).LT.0.0) y(S)=y(S)+MS
      ENDDO
      t=t+2*dt
      END
C
C ------------ Random number generator from Press et al -----------------
C
      function ran1(idum)
c     Returns a uniform deviate between 0.0 and 1.0. Set IDUM
c     to any negative value to initialise or reinintialise
c     the sequence
c
      integer j
      dimension r(97)
      parameter (m1=259200,ia1=7141,ic1=54773,rm1=1.0/m1)
```

```
      parameter (m2=134456,ia2=8121,ic2=28411,rm2=1.0/m2)
      parameter (m3=243000,ia3=4561,ic3=51349)
      save r,iff,ix1,ix2,ix3
      data iff /0/
c     Initialise on first call even if IDUM is not zero
      if (idum.lt.0.or.iff.eq.0) then
      iff=1
c     Seed the first routine
      ix1=mod((ic1-idum),m1)
      ix1=mod((ia1*ix1+ic1),m1)
c     and use it to seed the second
      ix2=mod(ix1,m2)
      ix1=mod((ia1*ix1+ic1),m1)
c     and the third routines
      ix3=mod(ix1,m3)
c     Fill the table with sequential uniform deviates generated
c     by the first two routines
      do 11 j=1,97
      ix1=mod((ia1*ix1+ic1),m1)
      ix2=mod((ia2*ix2+ic2),m2)
c     Low and high order pieces combined here
      r(j)=(float(ix1)+float(ix2)*rm2)*rm1
   11 continue
      idum=1
      endif
c     Except when initialising this is where we start.
c     Generate the next number for each sequence
      ix1=mod((ia1*ix1+ic1),m1)
      ix2=mod((ia2*ix2+ic2),m2)
      ix3=mod((ia3*ix3+ic3),m3)
c     Use the third sequence to get an integer between 1 and 97
      j=1+(97*ix3)/m3
      if (j.gt.97.or.j.lt.1) write (*,*) ‘ failure in j’
c     Return that table entry
      ran1=r(j)
c     and refill it
      r(j)=(float(ix1)+float(ix2)*rm2)*rm1
      return
      end
```