Appendix A

A Simple N-Body Integrator

N-body codes used in research run to many thousands of lines and hundreds of subroutines. The two main collisional codes/packages are

1) *Starlab* See `http://www.manybody.org`
2) *NBODYx* A series of N-body codes of differing sophistication and adapted to different hardware. See `ftp://ftp.ast.cam.ac.uk/pub/sverre/nbody6/`

The codes that follow are *much* more modest. We give both Matlab and C codes, but not a `FORTRAN` code (for which see Binney & Tremaine 1987).

Matlab

This code is presented mainly because of the integrated graphics capabilities of Matlab (Fig.1). To run the code, enter 'nbody(25,100,10)' in a matlab session, assuming that the following file is in the matlab search path. The input data are the number of particles, the number of outputs and the end time, respectively.

The first function initialises, runs at intervals of length `dt` (collecting data in `y`), and outputs a final result.

```
function nbody(n,nout,tmax)
cpu = cputime;dt=tmax/nout;t=0;
[h,x,xdot,f,fdot,step,tlast,m,y] = initialise(n);
nsteps=0;
while t<tmax
[x,xdot,f,fdot,step,tlast,m,t,nsteps] = ...
hermite(x,xdot,f,fdot,step,tlast,m,t,t+dt,nsteps);
```

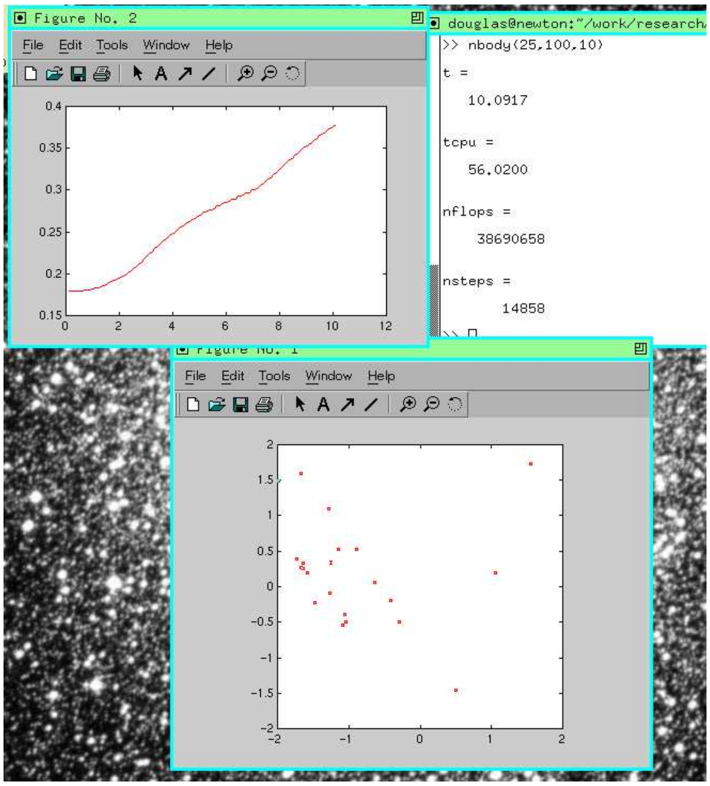

Fig. 1 Screen view after running the Matlab code.

```
[y]=runtime_output(h,x,y,t,n);
end
final_output(t,cpu,nsteps,y)
```

This function initialises various scalars, chooses n positions and velocities with a uniform distribution inside spheres whose radii are chosen to give N-body units approximately. It then sets masses, and times when particles were most recently updated (i.e. 0, initially). Forces, force derivatives and initial timesteps are selected (cf. Makino & Aarseth 1992). Finally a plot window for showing a movie is initialised.

```
function [h,x,xdot,f,fdot,step,tlast,m,y] = ...
initialise(n)
flops(0);y(1)=0;figure(1)
x = uniform(n,6/5);
xdot = uniform(n,sqrt(5/6));
m = ones(1,n)./n;
tlast = zeros(1,n);
for i = 1:n
```

```
[f(1:3,i),fdot(1:3,i)] = ffdot(x,xdot,m,i);
step(i) = 0.01*sqrt(sum(f(1:3,i).^2)/sum(fdot(1:3,i).^2));
end
h=plot(x(1,:),x(2,:),'o');
axis([-2 2 -2 2])
axis square
grid off
set(h,'EraseMode','xor','MarkerSize',2)
```

```
function [x] = uniform(n,a)
r = a*rand(1,n).^(1/3);
cos_theta = 2*rand(1,n)-1;
sin_theta = sqrt(1-cos_theta.^2);
phi = 2*pi*rand(1,n);
x(1,1:n) = r.*sin_theta.*cos(phi);
x(2,1:n) = r.*sin_theta.*sin(phi);
x(3,1:n) = r.*cos_theta;
```

Particles are updated in this function. In the main loop, the next particle for updating is found (`i`), positions and velocities of all particles are predicted, and force and force derivative on `i` are computed by a call to `ffdot` (see below). Higher order Hermite corrections `a2` and `a3` are computed and the corrections applied. Finally the next time step is estimated and some variables are updated.

```
function [x,xdot,f,fdot,step,tlast,m,t,nsteps] = ...
hermite (x, xdot, f, fdot, step, tlast, m, t, tend,nsteps)
n = size(m,2);
while t < tend
[t,i] = min(step + tlast);
dt = t - tlast;
dts = dt.^2;
dtc = dts.*dt;
dt3 = dt(ones(1,3),:);
dts3 = dts(ones(1,3),:);
dtc3 = dtc(ones(1,3),:);
xtemp = x + xdot.*dt3 + f.*dts3/2 + fdot.*dtc3/6;
xdottemp = xdot + f.*dt3 + fdot.*dts3/2;
[fi,fidot] = ffdot(xtemp,xdottemp,m,i);
a2 = (-6*(f(1:3,i) - fi) - step(i)*(4*fdot(1:3,i) + ...
2*fidot))/step(i)^2;
a3 = (12*(f(1:3,i) - fi) + 6*step(i)*(fdot(1:3,i) + ...
fidot))/step(i)^3;
x(1:3,i) = xtemp(1:3,i) + step(i)^4*a2/24 + ...
step(i)^5*a3/120;
```

```
xdot(1:3,i) = xdottemp(1:3,i) + step(i)^3*a2/6 + ...
step(i)^4*a3/24;
f(1:3,i) = fi;
fdot(1:3,i) = fidot;
modfi = sum(fi.^2);
modfidot = sum(fidot.^2);
moda2 = sum(a2.^2);
moda3 = sum(a3.^2);
step(i) = min(1.2*step(i),sqrt(0.02*(sqrt(modfi*moda2) ...
+ modfidot)/(sqrt(modfidot*moda3) + moda2)));
tlast(i) = t;
nsteps = nsteps + 1;
end

function [fi,fidot] = ffdot(x,xdot,m,i)
n = size(m,2);
j = [1:i-1,i+1:n];
rij = x(:,i)*ones(1,n-1) - x(:,j);
rijdot = xdot(:,i)*ones(1,n-1) - xdot(:,j);
r2 = sum(rij.^2);
rrdot = sum(rij.*rijdot);
r3 = r2.^1.5;
r5 = r3.*r2;
fi = - rij*(m(j)./r3)';
fidot = - rijdot*(m(j)./r3)' + 3*rij*(rrdot.*m(j)./r5)';
```

This customisable function updates the movie and collects data for plotting the rms distance of all particles against time.

```
function [y]=runtime_output(h,x,y,t,n)
set(h,'XData',x(1,:),'YData',x(2,:));
drawnow
m = size(y,1);
y(m+1,1) = sqrt(sum(sum(x(:,:).^2)))/n;
y(m+1,2) = t;

function final_output(t,cpu,nsteps,y)
t
tcpu=cputime-cpu
nflops=flops
nsteps
figure(2)
plot(y(2:end,2),y(2:end,1),'-')
```

C

The following code lacks comments, but is structured like the preceding matlab code, so that the comments there should be helpful. It can be found at http://www.manybody.org.**to be done**

The code may be compiled with `gcc`* `nbody.c -o nbody -lm`, and run with `nbody 25 100 10` (say), where the data are the number of stars, the number of outputs and the end time. It runs about six times faster than the Matlab code. No movie is produced. To plot the accumulated data `y`, `gnuplot` (http://www.gnuplot.org/) could be used.

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <time.h>
#include <assert.h>
#define NMAX 1024
#define NDIM 3
#define loop(idx,last) for (idx = 0; idx < last ; idx++)
#define min(x, y) (x<y?x:y)
typedef double real;
static int i, j, k;

void uniform(int n, real a, real temp[NMAX][NDIM]) {
real r, cos_theta, sin_theta, phi;
loop(i, n) {
r = a*pow(drand48(), 1./3.);
cos_theta = 2.*rand()/((real)RAND_MAX+1)-1;
sin_theta = sqrt(1-pow(cos_theta, 2));
phi = 2*M_PI*rand()/((real)RAND_MAX+1);
temp[i][0] = r*sin_theta*cos(phi);
temp[i][1] = r*sin_theta*sin(phi);
temp[i][2] = r*cos_theta;
}
}

void ffdot(real x[NMAX][NDIM], real xdot[NMAX][NDIM],
   real m[ ], int i, int n, real fi[ ], real fidot[ ]) {
real rij[NDIM], rijdot[NDIM], r2, r3, r5, rrdot;
loop(k, NDIM) { fi[k] = fidot[k] = 0;}
loop(j, n) {
if (j!=i) {
```

* We used version 2.95 on a Sun running Solaris 2.7, for example.

```
r2 = rrdot = 0;
loop(k, NDIM) {
rij[k] = x[i][k] - x[j][k];
rijdot[k] = xdot[i][k] - xdot[j][k];
r2 += pow(rij[k],2);
rrdot += rij[k]*rijdot[k];
}
r3 = pow(r2,1.5);
r5 = r3*r2;
assert(r3!=0&&r5!=0);
loop(k, NDIM) {
fi[k] = fi[k] - m[j]*rij[k]/r3;
fidot[k] = fidot[k] - m[j]*(rijdot[k]/r3 -
   3*rrdot*rij[k]/r5);
}
}
}
}

int hermite(real x[NMAX][NDIM], real xdot[NMAX][NDIM],
   real f[NMAX][NDIM], real fdot[NMAX][NDIM],
   real step[ ], real tlast[ ], real m[ ], real *t,
   real tend, int n)
{
real xtemp[NMAX][NDIM], xdottemp[NMAX][NDIM], fi[NDIM],
   fidot[NDIM];
real a2[NDIM], a3[NDIM], modfi, modfidot, moda2, moda3;
real tmin, dt, dts, dtc, dnmtr, temp;
int imin, nsteps;
assert(n<=NMAX);
nsteps=0;
do {
tmin = pow(10,6);
imin = 0;
loop(i, n) {
if (step[i] + tlast[i] < tmin) {
tmin = step[i] + tlast[i];
imin = i;
}
}
assert(imin>=0);
*t = tmin;
i = imin;
```

```
loop(j, n) {
dt = *t - tlast[j];
dts = pow(dt, 2);
dtc = dts*dt;
loop(k, NDIM) {
xtemp[j][k] = x[j][k] + xdot[j][k]*dt + f[j][k]*dts/2
+ fdot[j][k]*dtc/6;
xdottemp[j][k] = xdot[j][k] + f[j][k]*dt +
   fdot[j][k]*dts/2;
}
}
ffdot(xtemp,xdottemp,m,i,n,fi,fidot);
modfi = modfidot = moda2 = moda3 = 0;
assert(step[i]!=0);
loop(k, NDIM) {
a2[k] = (-6*(f[i][k] - fi[k]) - step[i]*(4*fdot[i][k]
+ 2*fidot[k]))/pow(step[i],2);
a3[k] = (12*(f[i][k] - fi[k]) + 6*step[i]*(fdot[i][k]
+ fidot[k]))/pow(step[i],3);
x[i][k] = xtemp[i][k] + pow(step[i],4)*a2[k]/24
+ pow(step[i],5)*a3[k]/120;
xdot[i][k] = xdottemp[i][k] + pow(step[i],3)*a2[k]/6
+ pow(step[i],4)*a3[k]/24;
f[i][k] = fi[k];
fdot[i][k] = fidot[k];
modfi += pow(fi[k],2);
modfidot += pow(fidot[k],2);
moda2 += pow(a2[k],2);
moda3 += pow(a3[k],2);
}
dnmtr = moda2 + sqrt(modfidot*moda3);
assert(dnmtr!=0);
temp = sqrt(0.02*(sqrt(modfi*moda2) + modfidot)/dnmtr);
step[i] = min(1.2*step[i], temp);
tlast[i] = *t;
nsteps++;
}
while(*t<tend);
return nsteps;
}
```

```
void initialise(int n, real x[NMAX][NDIM],
   real xdot[NMAX][NDIM], real f[NMAX][NDIM],
   real fdot[NMAX][NDIM], real step[NMAX],
   real tlast[NMAX], real m[ ]) {
real fi[NDIM],fidot[NDIM], radius, fmod, fdotmod;
radius = 6./5.;
uniform(n,radius,x);
radius = sqrt(5./6.);
uniform(n,radius,xdot);
loop(i, n) { m[i] = 1./(real)n;}
loop(i, n) {
tlast[i] = 0;
ffdot(x,xdot,m,i,n,fi,fidot);
loop(k, NDIM) {
f[i][k] = fi[k];
fdot[i][k] = fidot[k];
}
fmod = fdotmod = 0;
loop(k, NDIM) {
fmod += pow(fi[k],2);
fdotmod += pow(fidot[k],2);
}
step[i] = 0.01*sqrt(fmod/fdotmod);
}
}

void runtime_output(real x[NMAX][NDIM], real t,
   long int n){
real ss=0;
loop (i,n) ss += x[i][0]*x[i][0] + x[i][1]*x[i][1] +
   x[i][2]*x[i][2];
printf("%lf %lf\n",t,sqrt(ss));
}

void final_output(real t, long int cpu, long int nsteps,
   long int noutp){
printf("\n#%lf, %ld, %ld\n",
time, (clock()-cpu)/CLOCKS_PER_SEC, nsteps);
}

void main(int argc, char **argv) {
real x[NMAX][NDIM], xdot[NMAX][NDIM], f[NMAX][NDIM],
   fdot[NMAX][NDIM];
real step[NMAX], tlast[NMAX], m[NMAX], t, dt;
```

```
long int j, i, n, nsteps, noutp, cpu;
cpu = clock();
printf("%ld\n",cpu);
srand(cpu);
assert(argc==4);
n = atoi(argv[1]);
noutp = atoi(argv[2]);
dt = atof(argv[3])/(real)noutp;
t = 0;
initialise(n, x, xdot, f, fdot, step, tlast, m);
nsteps=0;
loop(j, noutp) {
nsteps += hermite(x, xdot, f, fdot, step, tlast, m, &t,
   t+dt, n);
runtime_output(x,t,n);
}
final_output(t,cpu,nsteps,noutp);
}
```

Problems

1) Given acceleration a and its derivative at the beginning and end (subscripts 0 and 1, respectively) of a time step of length h, obtain the approximate formulae

$$a_0^{(2)} = \frac{-6(a_0 - a_1) - h(4\dot{a}_0 + 2\dot{a}_1)}{h^2}$$
$$a_0^{(3)} = \frac{12(a_0 - a_1) + 6h(\dot{a}_0 + \dot{a}_1)}{h^3}$$

for the second and third derivatives at the start of the time step. (The Hermite integrator first takes the step using terms in a Taylor series up to $\dot{a}_0$, i.e.

$$x_1' = x_0 + hv_0 + \frac{1}{2}h^2 a_0 + \frac{1}{6}h^3 \dot{a}_0$$
$$v_1' = v_0 + ha_0 + \frac{1}{2}h^2 \dot{a}_0,$$

computes a_1 and $\dot{a}_1$, and then adds the terms in $a_0^{(2)}$ and $a_0^{(3)}$, i.e.

$$x_1 = x_1' + \frac{1}{24}h^4 a_0^{(2)} + \frac{1}{120}h^5 a_0^{(3)}$$
$$v_1 = v_1' + \frac{1}{6}h^3 a_0^{(2)} + \frac{1}{24}h^4 a_0^{(3)}.)$$

2) Convert the code in Binney & Tremaine 1987 to the Hermite integrator.
3) Modify the initial conditions of either N-body code to integrate the 3-body problem with masses $m_i = 1$ and the following initial conditions:

$$\mathbf{r}_1 = (-0.97000436, 0.24308753, 0) = -\mathbf{r}_2,$$
$$\mathbf{v}_1 = (-0.46620368, -0.43236573, 0) = \mathbf{v}_2,$$
$$\mathbf{r}_3 = \mathbf{0}, \mathbf{v}_3 = -2\mathbf{v}_1$$

(cf. ch.17A, Fig.4).
4) At present the output routines use data corresponding to the last update time. Modify so that it uses extrapolated data (as in the hermite function/routine).
5) Perform the same kind of N-body calculation as the simple Matlab and C codes (with the same initial conditions) using starlab.