Homework 2

Rakesh Nath

February 19, 2013

Problem 1

(a)

The code for this problem is in the appendix. This code uses a radius of 1 unit between particles, and a mass of 1×10^2 for each particle. The code needed an initial kick which is defined for the leap frog method by giving two different initial positions to particles. The leap frog method solution for a time N has a general solution as

$$X_N = 2X_{N-1} - X_{N-2} + g_{N-1}dt^2$$

where $X_N = x\hat{i} + y\hat{j}$
The acceleration of gravity g_i for a mass m_i for N particles is given by

$$g_i = \Sigma_{j=0,i\neq j}^{j=N} \frac{m_j X_i - X_j}{X_{ij}^3}$$

where $X_{ij} = \sqrt{(x_i - xj)^2 + (y_i - y_j)^2}$ Fig(1) shows ten orbits and Fig(2) shows

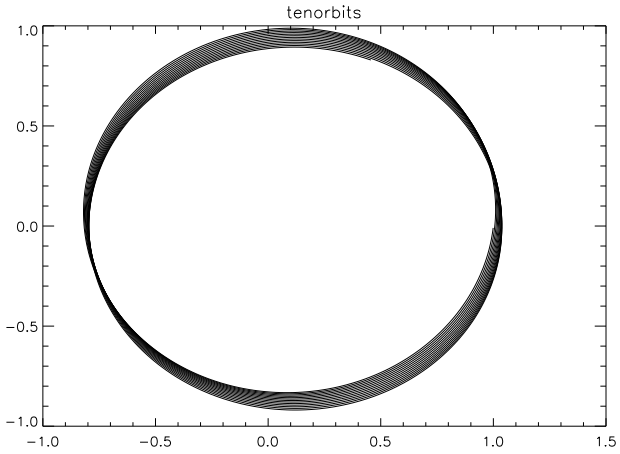

Figure 1: A figure of 10 orbits with a radius of 1

the energy evolution of the system.

(b)

The total energy is calculated by

$$E = PE + KE$$

$$PE = -\frac{1}{2} G \Sigma_{j=1,i\neq j}^{N} \frac{m_i m_j}{|r_i - r_j|}$$

$$KE = \frac{1}{2} \Sigma_{i=1}^{N} m_i (\frac{dr_i}{dt})^2$$

The Fig(3)shows energy evolution along the x-axis and y-axis.

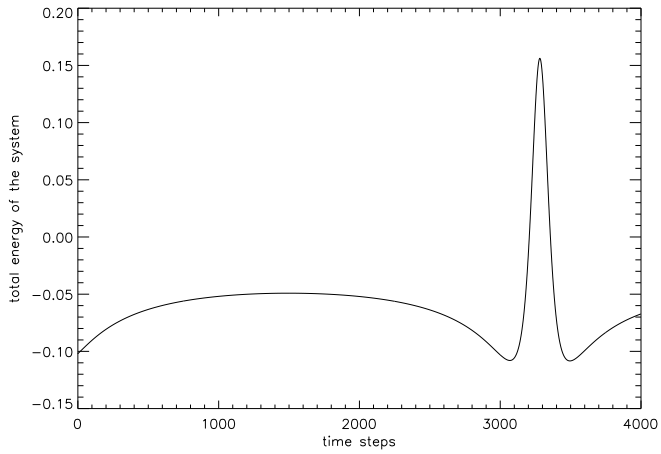

Figure 2: Energy evolution of the system with timesteps

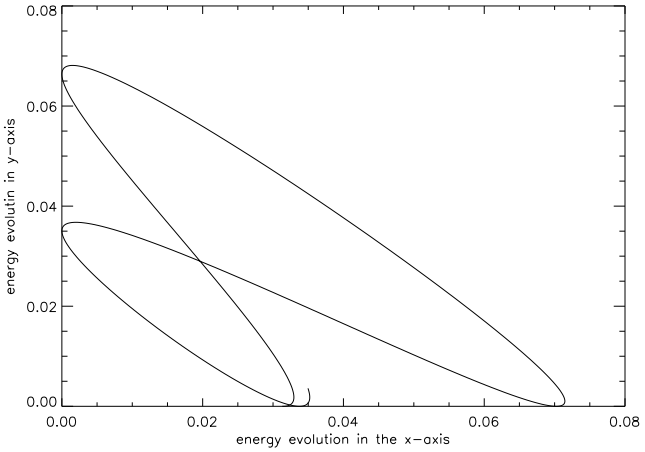

Figure 3: Plot of the energy variation along x and y axis

Problem 2

The orbits are much more accurately preserved in the leap frog method. All plots here were made for 200 time steps/orbit case. I did not have the time to execute the other case, in the sense that the time taken for the code to run was too long(nearly 30 minutes) I need to check why this is the case.

The orbits using Euler's method are displayed in Fig(5)

The energy variation according to the leapfrog scheme is shown in fig(6), it appears I made a bookkeeping error on this. This plot is wrong.

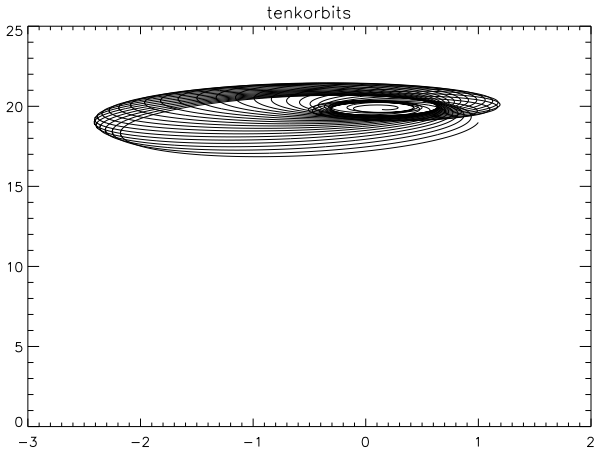

Figure 4: 10,000 orbits using leapfrog scheme

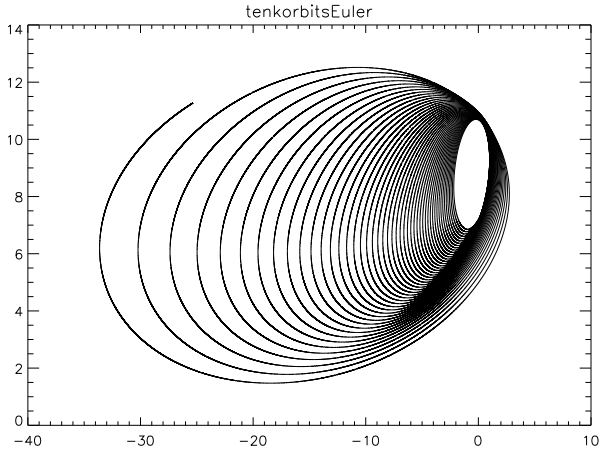

Figure 5: 10,000 orbits using Euler's scheme

Appendices

```
/*************************************************************************
Program name: Nbody
Description: This program does the main Nbody computations, it takes the
number of particles whose orbits is being calculated, calls the
acceleration subroutine to calculate the subsequent position using the
leap frog integration scheme.
Output:Returns 0 if successful
Input:none
*************************************************************************/
#include<stdio.h>
```

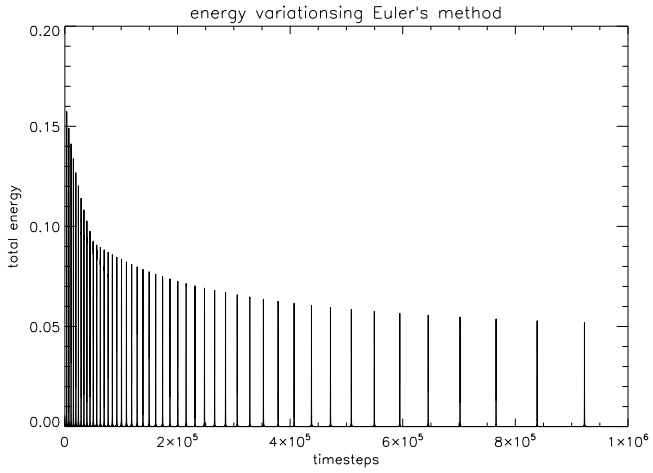

Figure 6: Energy variation in the Euler scheme

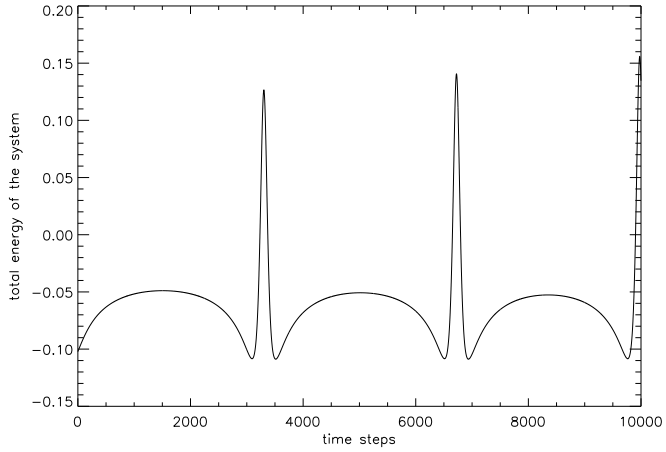

Figure 7: energy variation in the leap frog scheme

```
#include<file.h>
#include<math.h>
#include<omp.h>
#include<stdlib.h>
/*define the number of particle*/
#define N 2
/*define the mass of the system in grams*/
#define MASS 2e2
/*define the radius of the system in cm*/
#define RADIUS 1
/*define the number of orbits of the system*/
#define ORBIT 1
/*define the time resolution of the problem*/
```

```
#define TIME 1e-3
/*dimensions in the problem*/
#define DIM 2
double* fAccel(double*,double,double*,int);
double fEnergy(double,double,double);
double fPotential(double,double,double *,double*,int);
int main()
{
        double dTotTime,dParticleMass,dTimeinc;
        int iTimeSteps;
        dParticleMass=MASS/N;

        /*the actual variables are arrays because the acceleration due to
        gravity changes at every step and I would like to store that value
        in dAccelGravity.The position of each particle is stored as a
        2d array of the number of particles and dimensions*/
        double dAccelGravity[N][DIM]={0.0};
        FILE * fp1;
        FILE * fp2;
        FILE * accfp;
        FILE *vfp;
        FILE *exfp;
        FILE *eyfp;
        FILE *fpe;
        /*loop variables*/
        int i,j,k,iCount;
        /*the total time steps is how many time stamps the system will take,
          this will be higher if the time resolution of the system
         is smaller*/
                /*x and y positiions at time =0*/
        iTimeSteps=1e+9;
        double dPosition[DIM],dOtherPosition[DIM],*dAccel;
        /*The variables here are the current positions of
         * x and y and two positions in the past, the dimensions
         * are the number of particles N
         * ********************************/
        double dXcurr[N];
        double dXpast[N];
        double dXpastpast[N];

        double dEnergyx[N],dEnergyy[N];
        double dPotEnergy[N];
        double dYcurr[N],dYpast[N],dYpastpast[N];

/*initial x and y positions of the particles at time -1
         * for particle 0 and particle 1, they are placed in
         * opposite ends of the cooridinate system*/
        dXpastpast[0]=-RADIUS;
        dXpastpast[1]=RADIUS;
        dYpastpast[0]=RADIUS;
```

```
        dYpastpast[1]=-RADIUS;
        /*x and y positiions at time =0*/
        dXpast[0]=dXpastpast[0]+4.77e-4;
        dXpast[1]=dXpastpast[1]-4.77e-4;
        dYpast[0]=dYpastpast[0]+4.77e-3;//2.23e-4;
        dYpast[1]=dYpastpast[1]-4.77e-3;//2.224e-4;

        /*open the relevant file pointers*/

        fp1=fopen("particle1.dat","w+");
        fp2=fopen("particle2.dat","w+");
        accfp=fopen("accel.dat","w+");
        vfp=fopen("vel.dat","w+");
        exfp=fopen("energyx.dat","w+");
        eyfp=fopen("energyy.dat","w+");
        fpe=fopen("PE.dat","w+");
        /*the for loop that does the time integration and calculates
        particle positions for each time step*/
//#pragma omp reduction (+:dDummyReductionPosx,dDummyReductionPosy)
for(k=0;k<iTimeSteps;k++)
{/*for loop for the time step*/

   for(i=0;i<N;i++)
   {    /*for loop that cycles through all the particles in order to
         * calculate the effect of gravity
         * The next assignment is the x and y coordinates to
         * the dummy variable*/

            dPosition[0]=dXpast[i];
            dPosition[1]=dYpast[i];


        for(j=0;j<N;j++)
        {
                if(i==j)
                { /*the if construct to check for other particles*/
                        continue;
                }
        dOtherPosition[0]=dXpast[j];
        dOtherPosition[1]=dYpast[j];
dAccel=fAccel(dPosition,dParticleMass,dOtherPosition,DIM);
        }
        dAccelGravity[i][0]=dAccel[0];
        dAccelGravity[i][1]=dAccel[1];
        dEnergyx[i]=fEnergy(dAccelGravity[i][0],dParticleMass\
                        ,TIME);
        dEnergyy[i]=fEnergy(dAccelGravity[i][1],dParticleMass\
                        ,TIME);
        dPotEnergy[i]=fPotential(dParticleMass,dParticleMass,dPosition,\
                        dOtherPosition,DIM);
```

```
        /*Take the acceleration due to gravity and change the position
         * based on the previous 2 time stamp positions
         * the time increment is 10^-3 which is the time resolution*/
dXcurr[i]=2*dXpast[i]-dXpastpast[i]+dAccelGravity[i][0]*pow(TIME,2);
dYcurr[i]=2*dYpast[i]-dYpastpast[i]+ dAccelGravity[i][1]*pow((TIME),2);

        dXpastpast[i]=dXpast[i];
        dXpast[i]=dXcurr[i];
        dYpastpast[i]=dYpast[i];
        dYpast[i]=dYcurr[i];
/*this section just writes the position of the particle in
 * different files for different particles*/

if(i==1)
{
fprintf(fp1,"%5.10lf",dXcurr[i]);
fprintf(fp1,"\t%5.10lf\n",dYcurr[i]);
fprintf(fpe,"%5.10lf",dPotEnergy[i]);
}
else
{
fprintf(fp2,"%5.10lf",dXcurr[i]);
fprintf(fp2,"\t%5.10lf\n",dYcurr[i]);
fprintf(fpe,"\t%5.10lf\n",dPotEnergy[i]);
}
   }
        fprintf(exfp,"%5.10lf\t%lf\n",dEnergyx[0],dEnergyx[1]);
        fprintf(eyfp,"%lf\t%lf\n",dEnergyy[0],dEnergyy[1]);
        }
fflush(fp1);
fflush(fp2);
fclose(fp1);
fclose(fp2);
fclose(vfp);
fclose(exfp);
fclose(eyfp);
        return 0;
}

/*************************************************************************
Function Name: fAccel
Description: This function calculates the acceleration of a particle at a
certain distance and returns that values to integration routine
returns:double
Input:double,double
*************************************************************************/
#include<stdio.h>
#include<math.h>
#include<file.h>
/*define force softening*/
```

```
#define EPS 0
double *fAccel(double* r,double m,double* radius2,int dim)
        {
        /*define the variables radius from the center of mass of the particle'
         *the particle this is for the particle whose acceleration is being
         determined denoted by dRadius, the radius of the other particle
         wrt which the acceleration is being calculated dOtherParticle,
         the mass of the other particle dMass. This code will be run
         in a loop so all it will additively spit out the
         acceleration due to gravity */

                double* dRadius;
                dRadius=r;
                double dDistMag;
                double dMass=m;
                double dGrav[dim];
                double* dOtherParticle;
                dOtherParticle=radius2;
                int iDimensions=dim;
                int i;

for(i=0;i<iDimensions;i++)
        {
        dDistMag+=pow(dRadius[i]-dOtherParticle[i],2);
        }


/*calculate the acceleration due to gravity using
g=dMass*(dRadius-dOtherParticle)/(dRadius-dOtherParticle)^(3/2)
This is run N-1 times where N is the number of particles*/
for(i=0;i<iDimensions;i++)
{
        dGrav[i]=-dMass*(dRadius[i]-dOtherParticle[i])/pow(sqrt(dDistMag\
                                +pow(EPS,2)),3);
                }
                return dGrav;

        }

/*****************************************************************
 * Function name: fEnergy
 * Description:The function takes the input of acceleration and mass
 * converts it into velocity and returns the total kinetic energy of
 * the system.
 * Output:double
 * input: double,double
 * ***************************************************************/
#include<stdio.h>
#include<file.h>
#include<math.h>
```

```
#include<omp.h>
double fEnergy(double acc,double mass,double time)
        {
                double dAccl=acc;
                double dMass=mass;
                double dT=time;
                double dV=dT*dAccl;
                double dEnergy;
                dEnergy=dMass*pow(dV,2);
                return dEnergy;
        }

/*****************************************************************
Program name: Euler's method
Description: This program does the main Nbody computations using
euler's method,it takes the number of particles whose orbits
is being calculated, calls the
acceleration subroutine to calculate the subsequent position using the
Output:Returns 0 if successful
Input:none
*****************************************************************/
#include<stdio.h>
#include<file.h>
#include<math.h>
#include<omp.h>
#include<stdlib.h>
/*define the number of particle*/
#define N 2
/*define the mass of the system in grams*/
#define MASS 2e+2
/*define the radius of the system in cm*/
#define RADIUS 1
/*define the number of orbits of the system*/
#define ORBIT 1
/*define the time resolution of the problem*/
#define TIME 1e-3
/*dimensions in the problem*/
#define DIM 2
double *fAccel(double*,double,double*,int);
double fEnergy(double,double,double);
int main()
{
        double dTotTime,dParticleMass,dTimeinc;
        int iTimeSteps;
        dParticleMass=MASS/N;
        /*the actual variables are arrays because the acceleration due to
        gravity changes at every step and I would like to store that value
        in dAccelGravity.*/
        double dAccelGravity[N][DIM]={0.0};
        FILE * fp1;
```

```
FILE * fp2;
FILE * accfp;
FILE *ef1;
FILE *ef2;
/*loop variables*/
int i,j,k,iCount;
/*the total time steps is how many time stamps the system will take,
  this will be higher if the time resolution of the system
 is smaller*/
//time=ORBIT*dTotTime/TIME;
        /*x and y positiions at time =0*/

iTimeSteps=1e+6;

/*here an abscissa and an ordinate value for each particle is created
 * but then each value need not be stored, so only the initial values
 * are stored.
 */
/*dummy variable used inside the loop to pass values as
 * a vector to the acceleration subroutine*/

double dPosition[DIM],dOtherPosition[DIM],*dAccel;
/*variables to store the x and y values over varying time
 * steps of N particles*/
double dXpast[N],dYpast[N],dXcurr[N],dYcurr[N];
double dVxpast[N],dVypast[N],dVxcurr[N],dVycurr[N];
double dEnergyx[N],dEnergyy[N];


/*initial x and y positions of the particles at time -1
 * for particle 0 and particle 1, they are placed in
 * opposite ends of the cooridinate system*/
dXpast[0]=-RADIUS;
dXpast[1]=RADIUS;
dYpast[0]=RADIUS+10;
dYpast[1]=-RADIUS+10;
/*x and y positiions at time =0*/
dVxpast[0]=4.77e-1;
dVxpast[1]=-4.77e-1;

dVypast[0]=4.77;
dVypast[1]=-4.77;
/*      dPositionkx[1][0]=-9.996;
        dPositionkx[1][1]=9.996;
        dPositionky[1][0]=0.04;
        dPositionky[1][1]=-0.04;*/


fp1=fopen("particle1.dat","w+");
fp2=fopen("particle2.dat","w+");
```

```
        accfp=fopen("accel.dat","w+");
        ef1=fopen("energy1.dat","w+");
        ef2=fopen("energy2.dat","w+");
        /*the for loop that does the time integration and calculates
        particle positions for each time step*/

for(k=0;k<iTimeSteps;k++)
{/*for loop for the time step*/

    for(i=0;i<N;i++)
    {    /*for loop that cycles through all the particles in order to
          * calculate the effect of gravity
          * The next assignment is the x and y coordinates to
          * the dummy variable*/


            dPosition[0]=dXpast[i];
            dPosition[1]=dYpast[i];


        for(j=0;j<N;j++)
        {
                if(i==j)
                { /*the if construct to check for other particles*/
                        continue;
                }
        dOtherPosition[0]=dXpast[j];
        dOtherPosition[1]=dYpast[j];
dAccel=fAccel(dPosition,dParticleMass,dOtherPosition,DIM);

        }
        /*Take the acceleration due to gravity and change the position
         * based on the previous 2 time stamp positions
         * the time increment is 10^-2 which is the time resolution*/
//dPositionkx[k][i]=2*dPositionkx[k-1][i]-dPositionkx[k-2][i]+dAccelGravity[i]*
//dPositionky[k][i]=2*dPositionky[k-1][i]-dPositionky[k-2][i]+ dAccelGravity[i]
dAccelGravity[i][0]=dAccel[0];
dAccelGravity[i][1]=dAccel[1];

dEnergyx[i]=fEnergy(dAccelGravity[i][0],dParticleMass,TIME);
dEnergyy[i]=fEnergy(dAccelGravity[i][1],dParticleMass,TIME);


dXcurr[i]=dXpast[i]+(dVxpast[i]*TIME)+(dAccelGravity[i][0]*pow(TIME,2))/2;
dYcurr[i]=dYpast[i]+(dVypast[i]*TIME)+(dAccelGravity[i][1]*pow(TIME,2))/2;

dVxcurr[i]=dVxpast[i]+dAccelGravity[i][0]*pow(TIME,1);
dVycurr[i]=dVypast[i]+dAccelGravity[i][1]*pow(TIME,1);

/*this section just writes the position of the particle in
```

```
 * different files for different particles*/

if(i==1)
{
fprintf(fp1,"%e",dXpast[i]);
fprintf(fp1,"\t%e\n",dYpast[i]);
fprintf(accfp,"%e",dAccelGravity[i][0]);
fprintf(ef1,"%e",dEnergyx[i]);
fprintf(ef1,"\t%e\n",dEnergyy[i]);
}
else
{
fprintf(fp2,"%e",dXcurr[i]);
fprintf(fp2,"\t%e\n",dYcurr[i]);
fprintf(accfp,"\t%e\n",dAccelGravity[i][1]);
fprintf(ef2,"%e\t",dEnergyx[i]);
fprintf(ef2,"%e\n",dEnergyy[i]);
}
dXpast[i]=dXcurr[i];
dYpast[i]=dYcurr[i];
dVxpast[i]=dVxcurr[i];
dVypast[i]=dVycurr[i];
   }
        }
fclose(fp1);
fclose(fp2);
        return 0;
}

/*****************************************************************
 * Function Name:Potential.c
 * Description:It evaluates the potential energy of the particles
 * individually.
 * Input:double
 * Output:double
 * ***************************************************************/

#include<stdio.h>
#include<file.h>
#include<math.h>
#define G 6.67e-5
double fPotential(double m,double othermass,double* position,double* oposition,
                int dim)
        {
                double dMass=m;
                double dOtherMass=othermass;
                double *dPosition;
                dPosition=position;
                double *dOtherPosition;
                dOtherPosition=oposition;
```

```
        double dMagnitude;
        double dPE;
        int i,N=dim;
        for(i=0;i<N;i++)
        {
                dMagnitude+=pow(dPosition[i]-dOtherPosition[i],2);
        }
        dMagnitude=sqrt(dMagnitude);
        dPE=-(0.5)*G*dMass*dOtherMass/dMagnitude;//evaluation of PE
        return dPE;
}
```