Computational Physics - Project 5

Johannes Scheller (candidate no. 71), Vincent Noculak (candidate no. 22)
Lukas Powalla (candidate no. 67), Richard Asbah (candidate no. 50)

December 11, 2015

Contents

1 Execution

In order to analyse our data we need to find the potential energy and the kinetic energy at a given time t. Down here is the void function *KinpotEnergy* which calculates both the kinetic and potential energy for each particle. When ever this function is called in our RK4 or verlet method it calculates these values for one time step.

```
kineticEnergy <vector> //where we store each particle kinetic energy
potentialEnergy <vector> //where we store each particle potential energy

theTotalEnergy // where we store the total energy for all particles
total_kin // where we store the total kinetic energy only for the particles in system
total_pot // where we store the total potential energy only for the particles in system
numplanetsInSystem // the number of particles still in the system
ergodic //the value of ergodic ratio
```

```
void solarsystem::kinPotEnergy(){

    kineticEnergy = new double[ this->numplanets];
    total_kin = 0.0;
    total_pot = 0.0;

    double potenial = 0.0;
    double totalKinetic = 0.0;
    double totalPotenial = 0.0;
    double velocitySquared= 0.0;

    //kineticEnergy = 1/2 * m * v^2
    for (int i = 0; i < this->numplanets;i++){
        velocitySquared = A(3*i,1)*A(3*i,1)+A(3*i+1,1)*A(3*i+1,1)+A(3*i+2,1)*A(3*i+2,1);
        kineticEnergy[i] = 0.5*planets[i].m*velocitySquared;

        totalKinetic += kineticEnergy[i];
    }

    //potineal energy U = -G Mm/r-----------------
    potentialEnergy = new double[ this->numplanets];
    double r;
    Mat<double> p = Mat<double>(this->numplanets,this->numplanets,fill::zeros); //matrix p will include

    for (int i = 0;i < this->numplanets; i++ ){
        for (int j = i+1; j < this->numplanets; j++){
            r = (A(3*i,0)- A(3*j,0))*(A(3*i,0)- A(3*j,0)) + (A(3*i+1,0)- A(3*j+1,0))*(A(3*i+1,0)- A(3*j
            r = sqrt(r);
            p(i,j)=-planets[i].m*planets[j].m*G/r;
            p(j,i)=p(i,j);
            totalPotenial += p(j,i);
        }
    }
    //calcuting the potenial energy for each planet from the marix P
    for(int i = 0; i < this->numplanets; i++){
        for(int j = 0; j < this->numplanets; j++){
            potenial += p(i,j);
        }
        potentialEnergy[i] = potenial;
        potenial = 0;
    }
    theTotalEnergy = totalKinetic+totalPotenial;

    //Virial analysis
     numplanetsInSystem = 0;
    //clac the energy of the bound system ! virial !!!
    for (int i = 0;i < this->numplanets; i++ ){
        if (( kineticEnergy[i]+potentialEnergy[i])<0.0){
            numplanetsInSystem += 1;
            total_kin += kineticEnergy[i];
            total_pot += potentialEnergy[i];
        }

    }
    total_pot = total_pot/2;
    ergodic = total_pot/total_kin;
}
```

In the first part of the function, we calculate the kinetic energy for each particle as $1\frac{1}{2}mv(t)^2$

In the second part, we calculate all the potentials between different particles in a matrix p in the following way:

$$p = \begin{bmatrix} 0 & \frac{-Gm_1m_0}{r} & \frac{-Gm_2m_0}{r} & \dots & \frac{-Gm_nm_0}{r} \\ \frac{-Gm_0m_1}{r} & 0 & \frac{-Gm_2m_1}{r} & \dots & \frac{-Gm_nm_1}{r} \\ \dots & \dots & \dots & \dots & \dots \\ \frac{-Gm_0m_n}{r} & \frac{-Gm_1m_n}{r} & \frac{-Gm_2m_n}{r} & \dots & 0 \end{bmatrix} \tag{1}$$

Afterwards, we perform a new for loop, where we calculate the potential energy for a particle i as the sum over al elements of the ith row, and finally calculate the total potential and kinetic of the system by summing only over the bounded particle by using an if test to get only the particles with negative total energy.

In the method *centermassfunction*, we calculate the centre of mass $\mathbf{r}_c$ of the system. We loop over all bounded particles $\mathbf{r}_c = \frac{\sum_{i=1}^{n} m_i * \mathbf{r}_i}{m_{\text{total}}}$

```
void solarsystem::centermassfunction(){
    centermass = new double[3]; //(centrMassX,centerMassY,centerMassZ)
    double centermassX = 0.0;
    double centermassY = 0.0;
    double centermassZ = 0.0;
    double totalmass = 0.0;
    for (int i = 0;i<this->numplanets;i++){
      if (( kineticEnergy[i]+potentialEnergy[i])<0.0){
          totalmass  += planets[i].m;
          centermassX += A(3*i,0)*planets[i].m;
          centermassY += A(3*i+1,0)*planets[i].m;
          centermassZ += A(3*i+2,0)*planets[i].m;
        }
  }
    centermassX = centermassX/totalmass;
    centermassY = centermassY/totalmass;
    centermassZ = centermassZ/totalmass;
    centermass[0] = centermassX;
    centermass[1] = centermassY;
    centermass[2] = centermassZ;
}
```

Here we have the radial density function which calculate the average distance and the standard deviation of the particles as well. To calculate the radial density we first introduce a radial distance vector which tells the radius for each particle from the center of mass. In the first part of this function we test if the particle is bounded and store the distance to the center of mass. If it's not bounded we just put a huge distance(200), so we don't include this later in the radial density. The next step is to calculate the radial density. We loop over each particle and check if it has a radius in the region r to $r + step$, then we divide by the volume of a sphere with radius $r + step$ minus a sphere with radius r to get the radial density.

```
 void solarsystem::radialDensity(){

     double* radialDistance = new double[this->numplanets];
     double x = 0;
     double y = 0;
     double z = 0;
     double averageDistance = 0.0;
     double Nplanets = 0;
     double standardDeviation = 0.0;

     //looping though all bonded planets and clac the radial distance
     for (int i = 0;i<this->numplanets;i++){
        if (kineticEnergy[i]+potentialEnergy[i]<0){

            x = centermass[0] - A(3*i,0);
            y = centermass[1] - A(3*i+1,0);
            z = centermass[2] - A(3*i+2,0);
            double k = x*x+y*y+z*z;

            radialDistance[i] = sqrt(k);
            averageDistance += radialDistance[i];
            Nplanets += 1.0;

        }else{
            radialDistance[i] = 200; // just a big number to avoid worng calc(this plant is not in the
        }
    }

    averageDistance = averageDistance/Nplanets;

     //standard deviation
     for (int i = 0;i<this->numplanets;i++){
        if (radialDistance[i] < 200){
            standardDeviation += (radialDistance[i] - averageDistance)*(radialDistance[i] - averageDist
        }
    }
     standardDeviation = sqrt(standardDeviation/Nplanets);

     cout<<"number␣of␣planet␣in␣the␣system␣:␣"<<Nplanets<<endl;
     cout<<"average␣distance" <<averageDistance<<endl;
     cout<<"standard␣deviation" <<standardDeviation<<endl;
     double m= 0.0;
     for (int i=0; i < this->numplanets;i++){
    m += planets[i].m;
    }
     cout<<m<<endl;

   double maxRadius = 10;
   double step = 0.5;
   double N = 0.0; // number of planet inside a givin radius.
   double volume = 0.0;
   int steps =  ((int)maxRadius/step + 1);
   double* density = new double[steps];
   int loopintger = 0;// just an intger for the loop;


     for (double r = step; r < maxRadius; r += step){
       for (int i = 0;i < numplanetsInSystem; i++){
           if((radialDistance[i]<r)&(radialDistance[i]>(r-step))){
               N +=1;
           }
         }
       volume = 4.0/3.0 * M_PI* (r)*(r)*(r);
       volume = volume - 4.0/3.0 * M_PI* (r -step)*(r -step)*(r-step);
       density[loopintger]= N/volume;
```

```
        N=0;
        loopintger++;
      }

    //write data to a folder
    ofstream radialDensityTxt;
    radialDensityTxt.open ("radialDensity.txt");
    loopintger=0;
    for (double r = step; r < maxRadius; r += step){
        radialDensityTxt<<r<<"          "<<density[loopintger]<<endl;
        loopintger++;
    }

     radialDensityTxt.close();
}
```

Number of particles	average distance	standard deviation
50	11.283	15.02875
100	11.2452	14.9623
200	9.81885	15.2545
400	6.56	11.6193
800	5.298	6.749

With N increasing number of particles, the average distance and the standard deviation of the particles decreases.