The Three-Body Problem

Jack Kraus
PHYS 510

The problem

- Famous for not having a closed-form analytical solution - one can’t reliably form an ansatz (judicious guess) solution to a three-body configuration.
- We can, however, work numerically to get a good approximation of what each solution should look like.

Equations of Motion

The mathematical statement of the three-body problem can be given in terms of the Newtonian equations of motion for vector positions $\mathbf{r_i} = (x_i, y_i, z_i)$ of three gravitationally interacting bodies with masses m_i:

$$\ddot{\mathbf{r}}_1 = -Gm_2\frac{\mathbf{r}_1 - \mathbf{r}_2}{|\mathbf{r}_1 - \mathbf{r}_2|^3} - Gm_3\frac{\mathbf{r}_1 - \mathbf{r}_3}{|\mathbf{r}_1 - \mathbf{r}_3|^3},$$

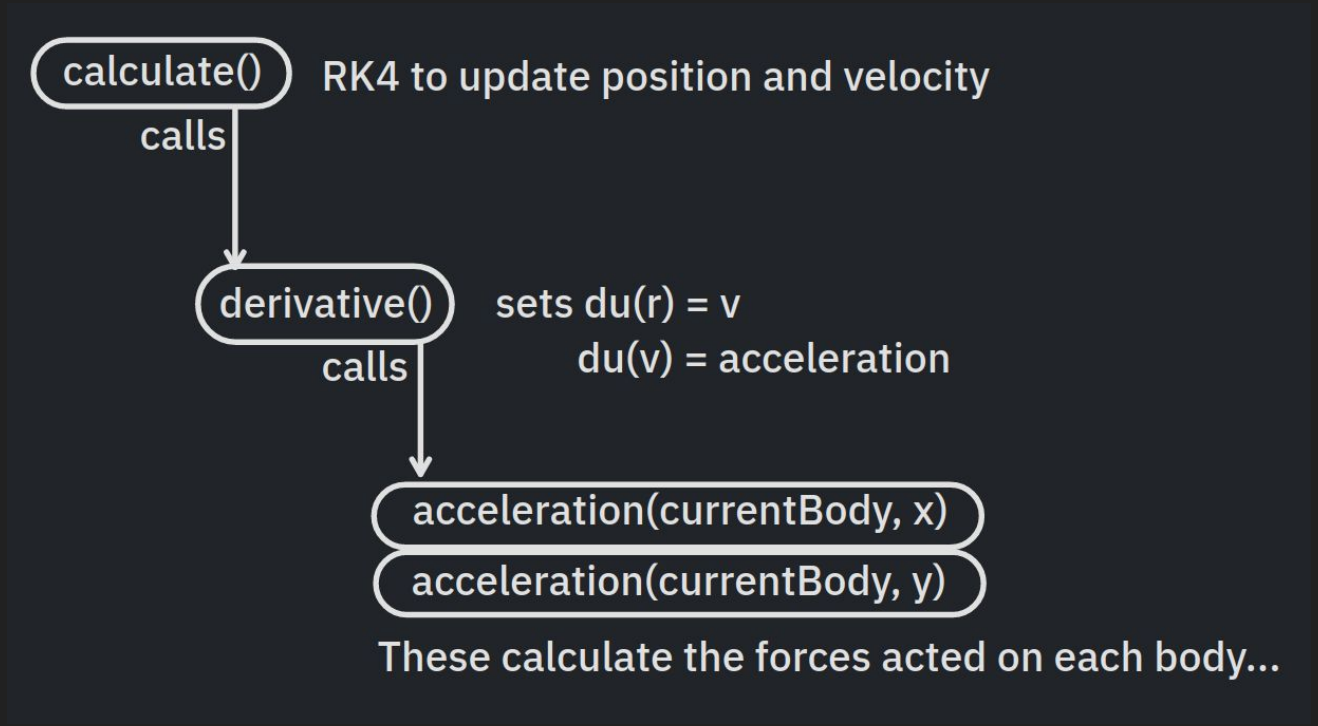
$$\ddot{\mathbf{r}}_2 = -Gm_3\frac{\mathbf{r}_2 - \mathbf{r}_3}{|\mathbf{r}_2 - \mathbf{r}_3|^3} - Gm_1\frac{\mathbf{r}_2 - \mathbf{r}_1}{|\mathbf{r}_2 - \mathbf{r}_1|^3},$$

$$\ddot{\mathbf{r}}_3 = -Gm_1\frac{\mathbf{r}_3 - \mathbf{r}_1}{|\mathbf{r}_3 - \mathbf{r}_1|^3} - Gm_2\frac{\mathbf{r}_3 - \mathbf{r}_2}{|\mathbf{r}_3 - \mathbf{r}_2|^3}.$$

Original strategy

```
//over each time step
 //TODO: 1. Calculate Forces from EOM
 //TODO: 2. Update Velocity
 //TODO: 3. Update Position
 //TODO: 4. Write out time, pos 1, pos 2, pos 3
```

More practical strategy



main(){

…

```
r=new point2D[num_bodies];
v=new point2D[num_bodies];
u=new double[4*N];
m=new double[num_bodies];


init(r,v,m,u); //initialize it first here

for(n=0;n<Nt;n++) {
    t=n*ht;
    init(r,v,m,u); //reinitilize it after every single time step
    calculate(t, u, m);
    //TODO: 4. Write out time, pos 1, pos 2, pos 3
    s = FloatToStr(t)+"\t"+ FloatToStr(u[0]) + "\t"+FloatToStr(u[1]) + "\t"
                          + FloatToStr(u[4]) + "\t"+FloatToStr(u[5]) + "\t"
                          + FloatToStr(u[8]) + "\t"+FloatToStr(u[9]) + "\n";
    FileWrite(f,s.c_str(),s.length());
}

FileClose(f);

//graph the posX vs posY Graph
gp->plotfile("3bp.txt","u 2:3 w l t 'Earth'");
gp->replotfile("3bp.txt","u 4:5 w l t 'Moon'");
gp->replotfile("3bp.txt","u 6:7 w l t 'Sun'");
gp->addcommand("reset");
gp->show();
```

…

return 0;

}

```
//RK4 to update position and velocity all at once
static void calculate(double h, double *u, double *m) {
    double a[4] = {h/2, h/2, h, 0}; // for RK4
    double b[4] = {h/6, h/3, h/3, h/6}; // for RK4
    double *u0, *ut;
    int uSize = sizeof(u);

    u0 = new double[uSize];
    ut = new double[uSize];

    int dimensionOfArray = uSize;

    for (int i = 0; i < dimensionOfArray; i++) {
        u0[i] = u[i]; // keep our initial value the same
        // if(i==0) printf("u[i] here is %f\n",u[i]); // FINE HERE
        ut[i] = 0.0; // we want to reset and previously existing value here
    }

    for (int j = 0; j < 4; j++) { // over the number of steps for RK4
        // need the derivatives for both position and velocity
        double *du = derivative(u,m);

        // goes to 4*numBodies = (12 for 3 bodies, 16 for 4 bodies)
        for (int i = 0; i < dimensionOfArray; i++) {
            // printf("du[i] here is %f\n",du[i]);
            u[i] = u0[i] + a[j]*du[i]; // initial value
            ut[i] = ut[i] + b[j]*du[i]; // time stepped value
        }
    }

    for (int i = 0; i < dimensionOfArray; i++) {
        u[i] = u0[i] + ut[i];
    }
}
```

Loop 1:

- initializing/resetting temp

Loop 2:

- obtain double array du from derivative()

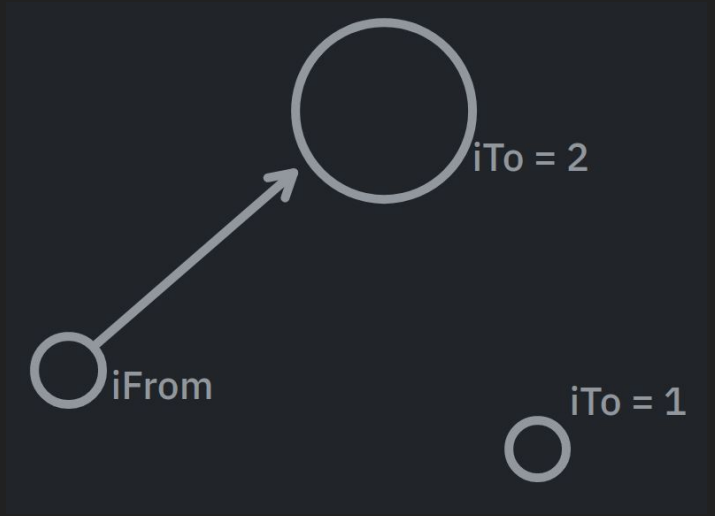
```
//---------------------------------------------------------------------------
//calculate the derivative

double * derivative(double *u, double *m) {
    double *du;
    du = new double[num_bodies * 4];

    // Loop through the bodies
    for (int iBody = 0; iBody < num_bodies; iBody++) {
        // Starting index for current body in the u array
        int bodyStart = iBody * 4;
        du[bodyStart + 0] = u[bodyStart + 2]; // dr_(bodyStart)_x = v_(bodyStart)_x
        du[bodyStart + 1] = u[bodyStart + 3]; // dr_(bodyStart)_y = v_(bodyStart)_y
        du[bodyStart + 2] = acceleration(iBody, 0, u, m); // Acceleration x
        du[bodyStart + 3] = acceleration(iBody, 1, u, m); // Acceleration y
        // printf("du[bodyStart + 2] here is %f\n",du[bodyStart + 3]);

    }
    return du;
}
```

For each body (iBody) set

- $du[r_x] = v_x$
- $du[r_y] = v_y$
- $du[v_x] = accel(iBody, x)$
- $du[v_y] = accel(iBody, y)$

accelerate() between the bodies

```
//-------------------------------------------------------------------------
//calculate the acceleration

//acceleration for each component, is called a total: 6 times for 3 bodies
double acceleration(int fromBody, int coord, double *u, double *m){
    double result = 0.0;
    int fromBodyStart = fromBody * 4; //time 4 because the index of the pos/velocity array holds 4 values per body
    double distX, distY, dist, overDist3;

    double G = 6.673e-11;
    // double G = 1.0;

    //loop through the bodies
    for(int iToBody = 0; iToBody < num_bodies; iToBody++){ //iterates through all of the bodies
        if(fromBody == iToBody) { continue; } // is the same, iterate to the next body
        int iToBodyStart = iToBody * 4;

        // Distance between the two bodies
        distX = u[iToBodyStart + 0] - u[fromBodyStart + 0]; //separation of each object in X
        distY = u[iToBodyStart + 1] - u[fromBodyStart + 1]; //separation of each object in Y

        dist = sqrt(distX*distX + distY*distY); //calculate the total distance between the two objects

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

        //okay this works
        overDist3 = 1/(dist*dist*dist); // 1/distance^3
        // printf("overDist3 = %.50f\n",overDist3);

        result += G*m[iToBody]*(u[iToBodyStart + coord] - u[fromBodyStart + coord])*overDist3; //the net force
    }

    // printf("acceleration = %f\n",result);
    return result;
}
```



accelerate() between the bodies

```
//-------------------------------------------------------------------------
//calculate the acceleration

//acceleration for each component, is called a total: 6 times for 3 bodies
double acceleration(int fromBody, int coord, double *u, double *m){
    double result = 0.0;
    int fromBodyStart = fromBody * 4; //time 4 because the index of the pos/velocity array holds 4 values per body
    double distX, distY, dist, overDist3;

    double G = 6.673e-11;
    // double G = 1.0;

    //loop through the bodies
    for(int iToBody = 0; iToBody < num_bodies; iToBody++){ //iterates through all of the bodies
        if(fromBody == iToBody) { continue; } // is the same, iterate to the next body
        int iToBodyStart = iToBody * 4;

        // Distance between the two bodies
        distX = u[iToBodyStart + 0] - u[fromBodyStart + 0]; //separation of each object in X
        distY = u[iToBodyStart + 1] - u[fromBodyStart + 1]; //separation of each object in Y

        dist = sqrt(distX*distX + distY*distY); //calculate the total distance between the two objects

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

        //okay this works
        overDist3 = 1/(dist*dist*dist); // 1/distance^3
        // printf("overDist3 = %.50f\n",overDist3);

        result += G*m[iToBody]*(u[iToBodyStart + coord] - u[fromBodyStart + coord])*overDist3; //the net force
    }

    // printf("acceleration = %f\n",result);
    return result;
}
```

calculates acceleration in coord={x,y}

For each body that isn't the current selected

- take the distance in x
- take the distance in y
- calculate ||r||^-3
- result += gravitational acceleration

```
//---------------------------------------------------------------------------
//calculate the derivative

double * derivative(double *u, double *m) {
    double *du;
    du = new double[num_bodies * 4];

    // Loop through the bodies
    for (int iBody = 0; iBody < num_bodies; iBody++) {
        // Starting index for current body in the u array
        int bodyStart = iBody * 4;
        du[bodyStart + 0] = u[bodyStart + 2]; // dr_(bodyStart)_x = v_(bodyStart)_x
        du[bodyStart + 1] = u[bodyStart + 3]; // dr_(bodyStart)_y = v_(bodyStart)_y
        du[bodyStart + 2] = acceleration(iBody, 0, u, m); // Acceleration x
        du[bodyStart + 3] = acceleration(iBody, 1, u, m); // Acceleration y
        // printf("du[bodyStart + 2] here is %f\n",du[bodyStart + 3]);

    }
    return du;
}
```

For each body (iBody) set

- $du[r_x] = v_x$
- $du[r_y] = v_y$
- $du[v_x] = a(iBody, x)$ ✅
- $du[v_y] = a(iBody, y)$ ✅

return du;

```
//RK4 to update position and velocity all at once
static void calculate(double h, double *u, double *m) {
    double a[4] = {h/2, h/2, h, 0}; // for RK4
    double b[4] = {h/6, h/3, h/3, h/6}; // for RK4
    double *u0, *ut;
    int uSize = sizeof(u);

    u0 = new double[uSize];
    ut = new double[uSize];

    int dimensionOfArray = uSize;

    for (int i = 0; i < dimensionOfArray; i++) {
        u0[i] = u[i]; // keep our initial value the same
        // if(i==0) printf("u[i] here is %f\n",u[i]); // FINE HERE
        ut[i] = 0.0; // we want to reset and previously existing value here
    }

    for (int j = 0; j < 4; j++) { // over the number of steps for RK4
        // need the derivatives for both position and velocity
        double *du = derivative(u,m);

        // goes to 4*numBodies = (12 for 3 bodies, 16 for 4 bodies)
        for (int i = 0; i < dimensionOfArray; i++) {
            // printf("du[i] here is %f\n",du[i]);
            u[i] = u0[i] + a[j]*du[i]; // initial value
            ut[i] = ut[i] + b[j]*du[i]; // time stepped value
        }
    }

    for (int i = 0; i < dimensionOfArray; i++) {
        u[i] = u0[i] + ut[i];
    }
}
```

Loop 1:

- initializing/resetting temp

Loop 2:

- obtain double array du[ ] from derivative() ✅

  Loop 2.5:

  - Update each position / velocity simultaneously from its initial position with u0[ ] (initial value) and ut[ ] (time-stepped), using the du[ ]

Loop 3

- Add the results of each element in u0[i] and ut[i] to get the true results.

main(){

…

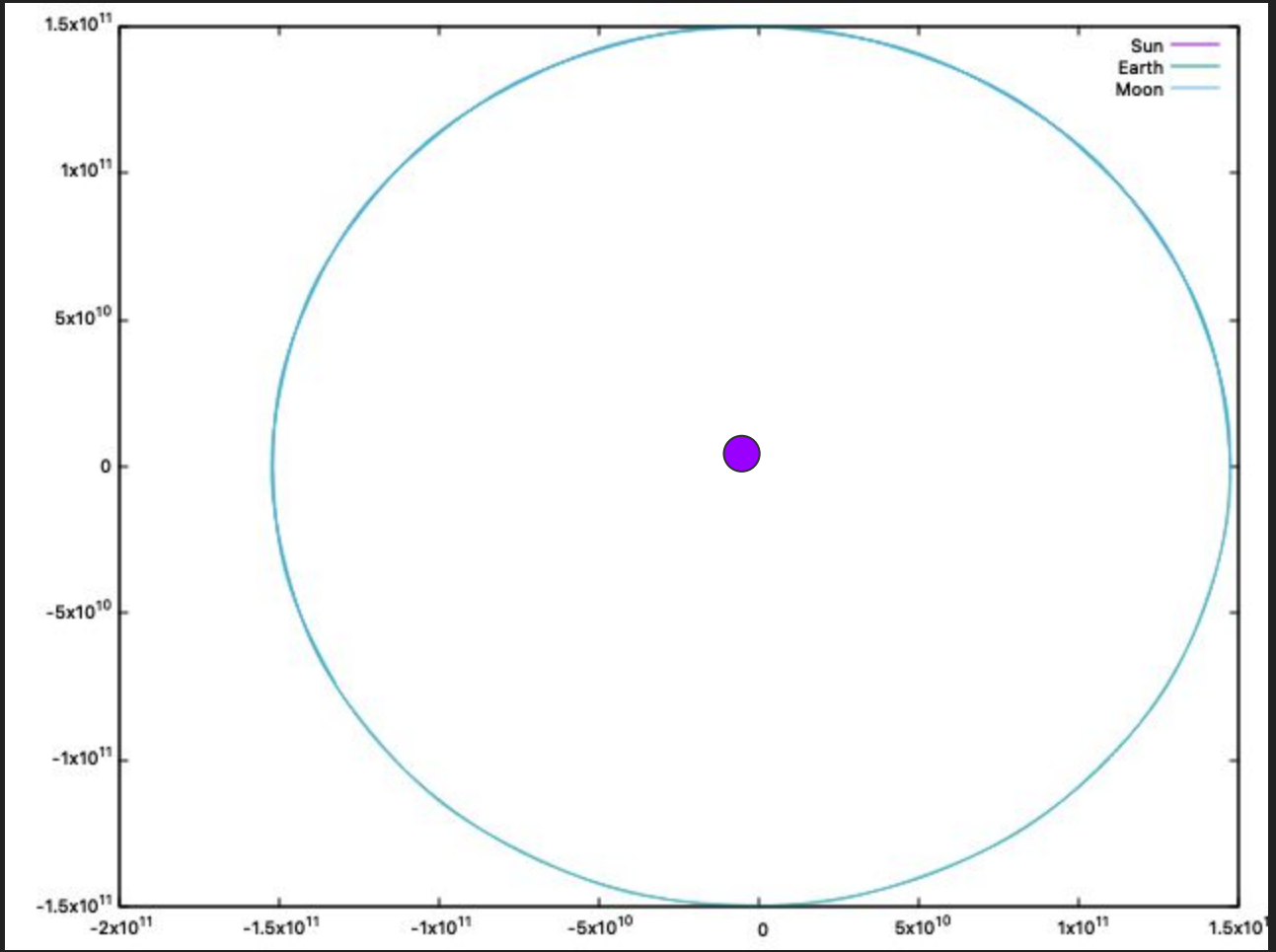
```
r=new point2D[num_bodies];
v=new point2D[num_bodies];
u=new double[4*N];
m=new double[num_bodies];


init(r,v,m,u); //initialize it first here

for(n=0;n<Nt;n++) {
    t=n*ht;
    init(r,v,m,u); //reinitilize it after every single time step
    calculate(t, u, m);
    //TODO: 4. Write out time, pos 1, pos 2, pos 3
    s = FloatToStr(t)+"\t"+ FloatToStr(u[0]) + "\t"+FloatToStr(u[1]) + "\t"
                          + FloatToStr(u[4]) + "\t"+FloatToStr(u[5]) + "\t"
                          + FloatToStr(u[8]) + "\t"+FloatToStr(u[9]) + "\n";
    FileWrite(f,s.c_str(),s.length());
}

FileClose(f);

//graph the posX vs posY Graph
gp->plotfile("3bp.txt","u 2:3 w l t 'Earth'");
gp->replotfile("3bp.txt","u 4:5 w l t 'Moon'");
gp->replotfile("3bp.txt","u 6:7 w l t 'Sun'");
gp->addcommand("reset");
gp->show();
```

…

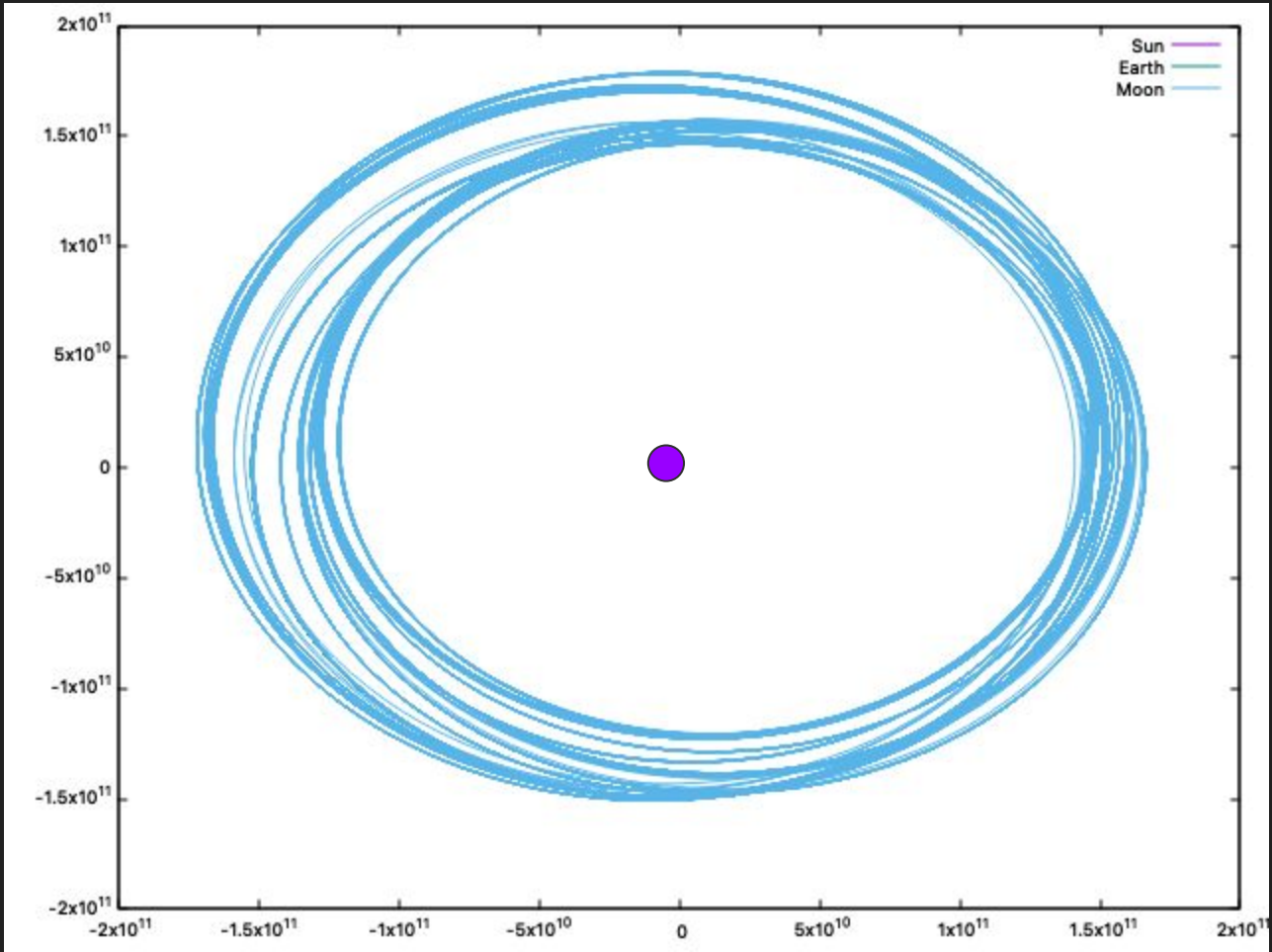
return 0;

}

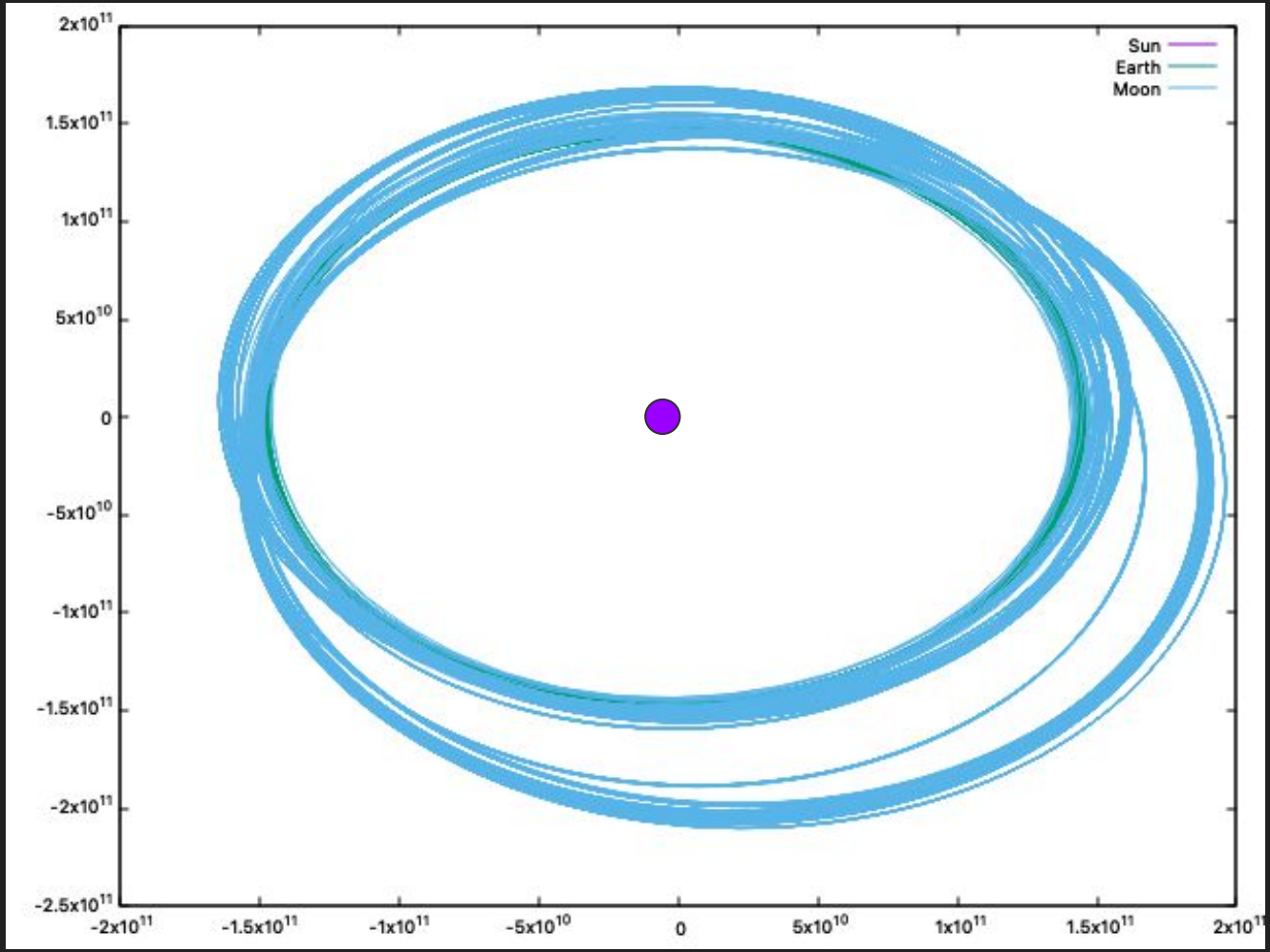
h = 0.1


Sun
Earth
Moon
1.5x10^11
1x10^11
5x10^10
0
-5x10^10
-1x10^11
-1.5x10^11
-2x10^11
-1.5x10^11
-1x10^11
-5x10^10
0
5x10^10
1x10^11

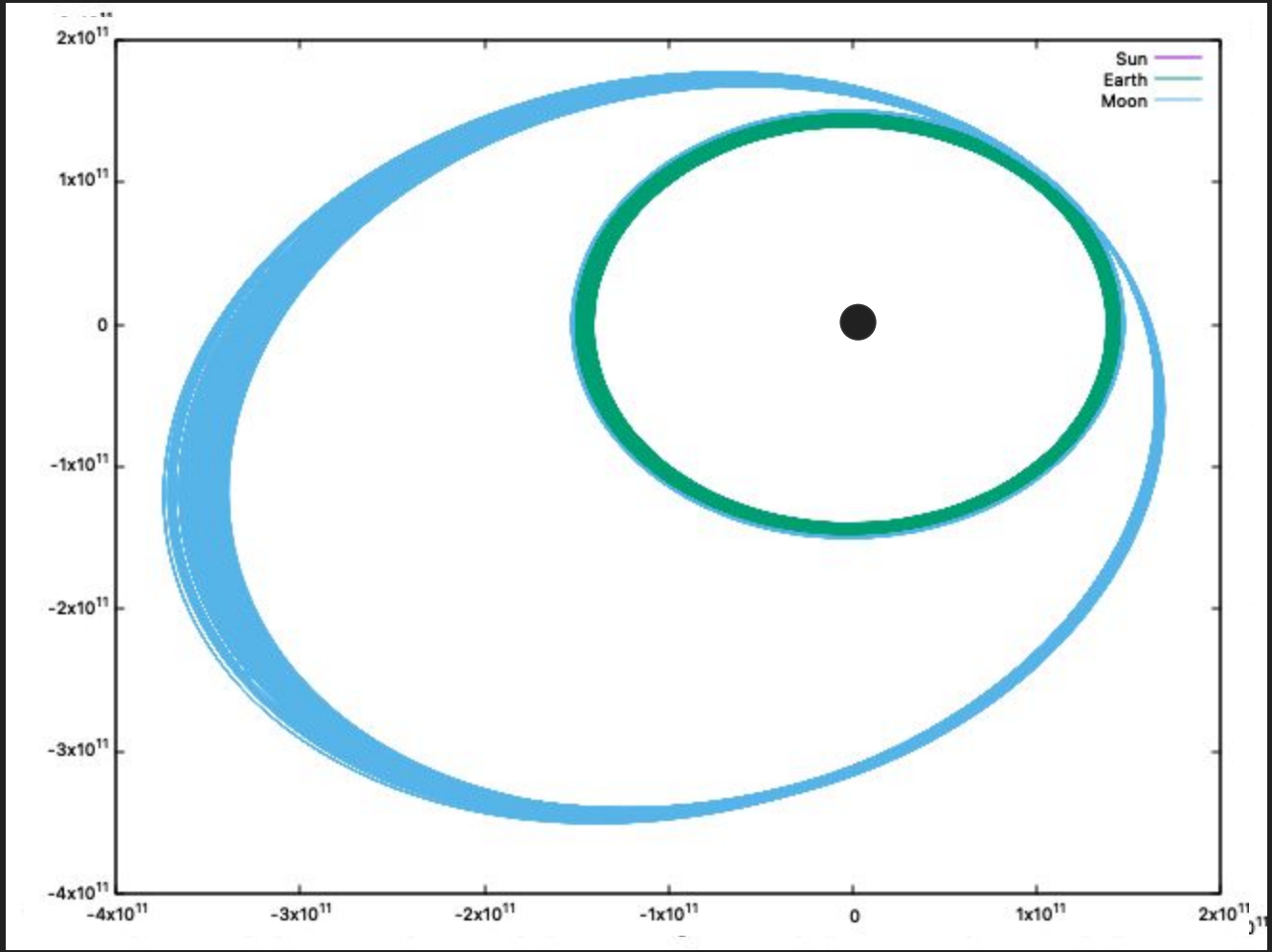
h = 5


Sun
Earth
Moon
2x10^11
1.5x10^11
1x10^11
5x10^10
0
-5x10^10
-1x10^11
-1.5x10^11
-2x10^11
-2x10^11
-1.5x10^11
-1x10^11
-5x10^10
0
5x10^10
1x10^11
1.5x10^11
2x10^11

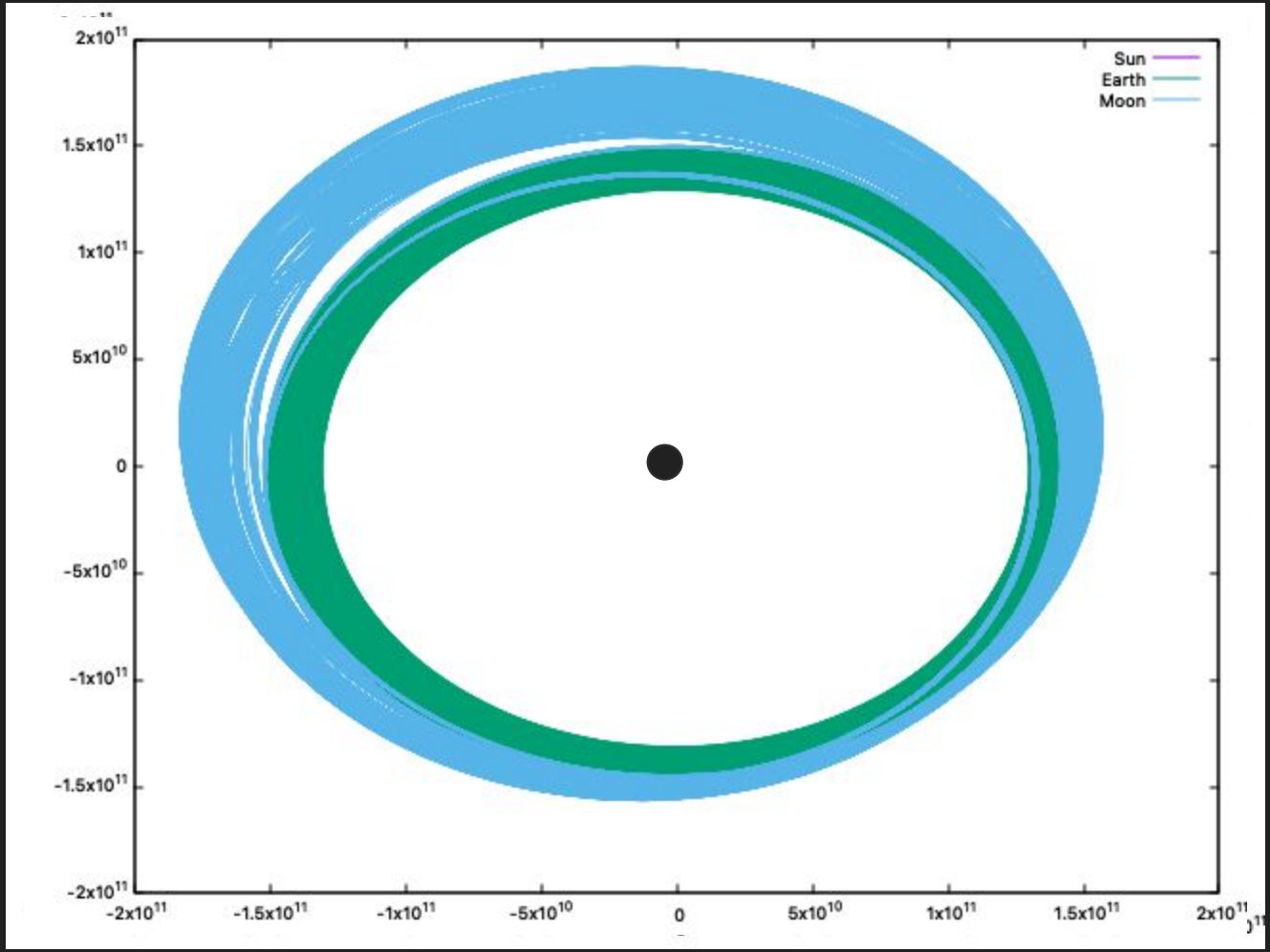
h = 10


Sun
Earth
Moon
2x10^11
1.5x10^11
1x10^11
5x10^10
0
-5x10^10
-1x10^11
-1.5x10^11
-2x10^11
-2.5x10^11
-2x10^11
-1.5x10^11
-1x10^11
-5x10^10
0
5x10^10
1x10^11
1.5x10^11
2x10^11

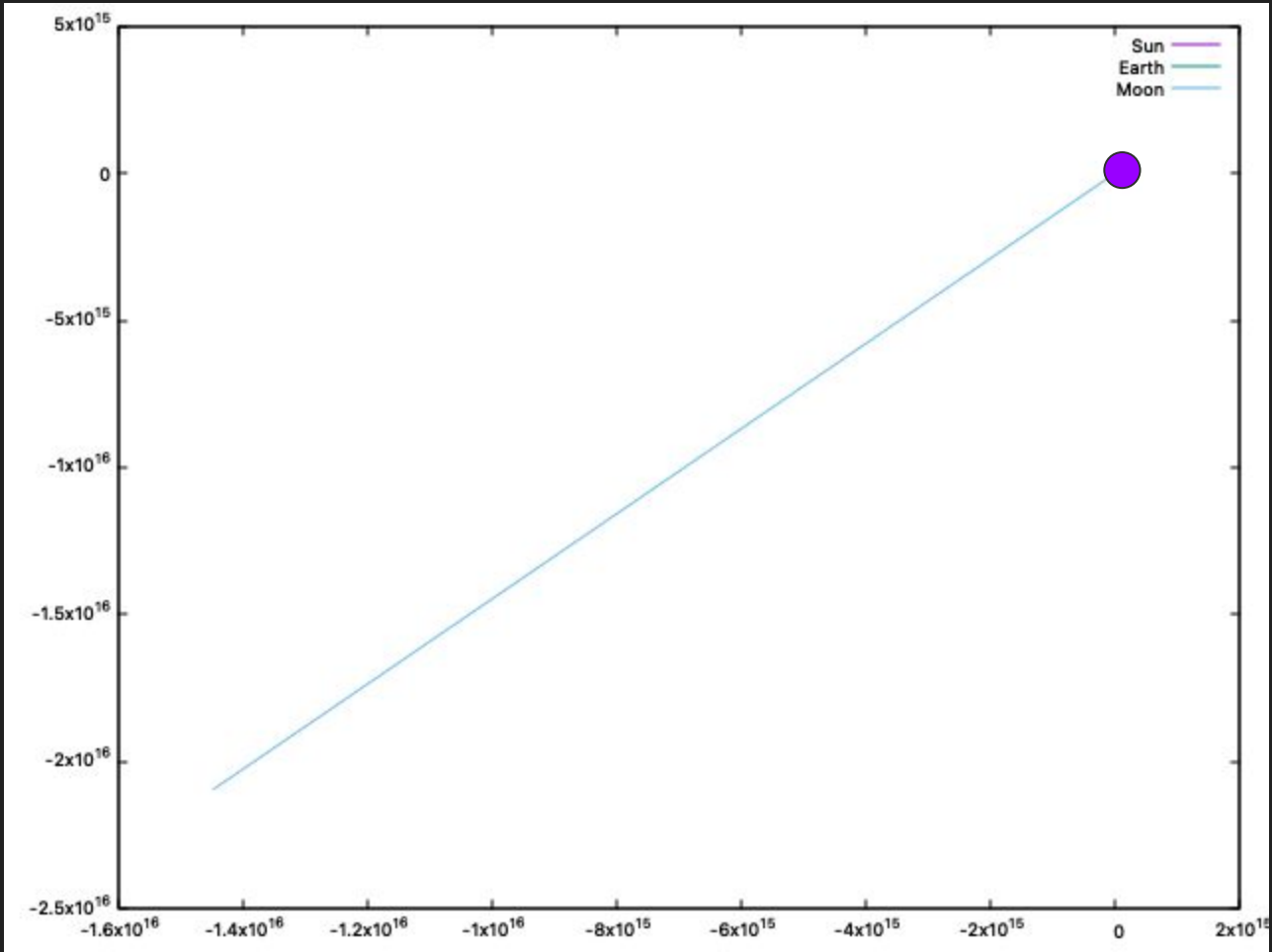
h = 11


Sun
Earth
Moon
2x10^11
1x10^11
0
-1x10^11
-2x10^11
-3x10^11
-4x10^11
-4x10^11
-3x10^11
-2x10^11
-1x10^11
0
1x10^11
2x10^11

h = 12


Sun
Earth
Moon
2x10^11
1.5x10^11
1x10^11
5x10^10
0
-5x10^10
-1x10^11
-1.5x10^11
-2x10^11
-2x10^11
-1.5x10^11
-1x10^11
-5x10^10
0
5x10^10
1x10^11
1.5x10^11
2x10^11

h = 13
Breaks!


Sun
Earth
Moon
5x10^15
0
-5x10^15
-1x10^16
-1.5x10^16
-2x10^16
-2.5x10^16
-1.6x10^16
-1.4x10^16
-1.2x10^16
-1x10^16
-8x10^15
-6x10^15
-4x10^15
-2x10^15
0
2x10^15

Problems

1. Time scales
    a. Need to figure out how to simulate a year in rotation
2. Length scales
    a. gnuplot skips from 0 to O(1e11 meters)
    b. This is fine when sim works properly
3. ~~Accurate simulations~~
    a. ~~Moon and Earth are engulfed by the Sun almost immediately, (ideally should be an orbit)~~
4. Large files, memory allocation issues.
    a. Output file gets into the GB, animating every 100 or 1000 events still takes a lot of processing power.
    b. **So** keep Nt smaller than 1e6

None of these are impossible, which is ✅ good ✅

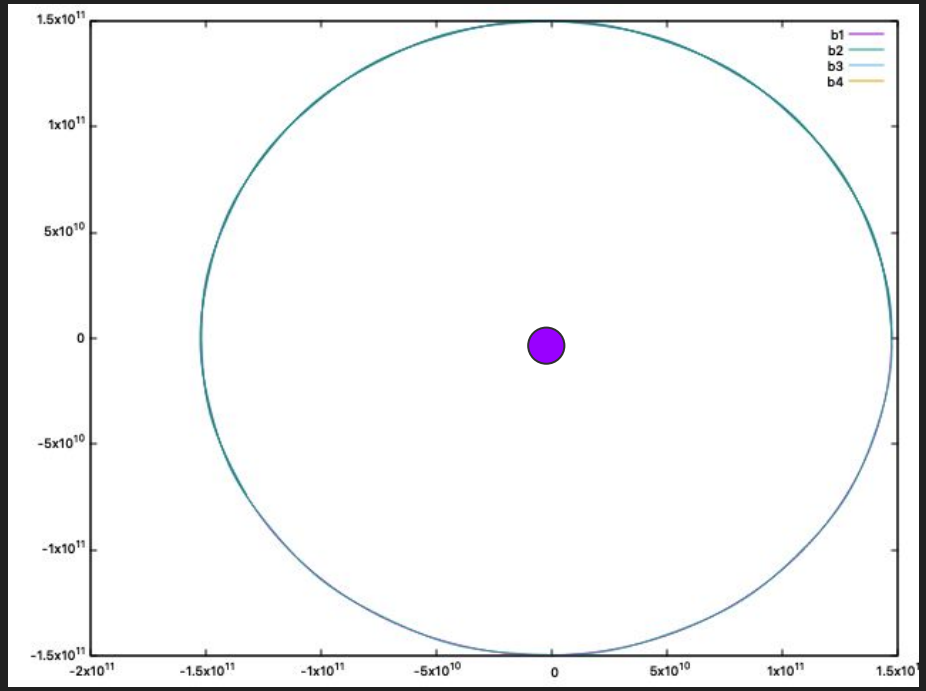
Errors (when Nt too large)

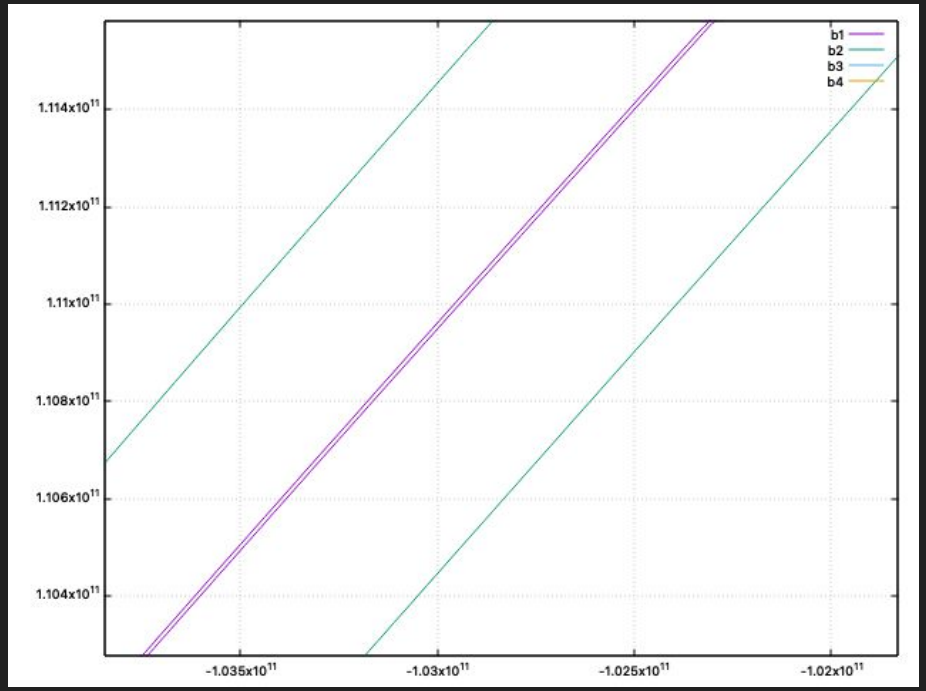
Yowza, that's a big file. Try again with a file smaller than 25MB.

```
jack@x1yoga:~/Desktop/510/three-body-problem$ ./3bp
Killed
jack@x1yoga:~/Desktop/510/three-body-problem$ g++ -o 3bp 3bp.cpp stringutils.cpp
 fileutils.cpp gnuplot.cpp
jack@x1yoga:~/Desktop/510/three-body-problem$ ./3bp
^C
jack@x1yoga:~/Desktop/510/three-body-problem$ g++ -o 3bp 3bp.cpp stringutils.cpp
 fileutils.cpp gnuplot.cpp
jack@x1yoga:~/Desktop/510/three-body-problem$ ./3bp
Killed
jack@x1yoga:~/Desktop/510/three-body-problem$ g++ -o 3bp 3bp.cpp stringutils.cpp
 fileutils.cpp gnuplot.cpp
jack@x1yoga:~/Desktop/510/three-body-problem$ ./3bp
Killed
jack@x1yoga:~/Desktop/510/three-body-problem$
```

```
jackraus three-body-problem-main >> g++ -o 4bp 4bp.cpp stringutils.cpp fileutils.cpp gnuplot.cpp
jackraus three-body-problem-main >> ./4bp
qt.qpa.fonts: Populating font family aliases took 49 ms. Replace uses of missing font family "Sans" with one
 that exists to avoid this cost.
4bp(10065,0x1e0f0db40) malloc: *** error for object 0x600001ce4120: pointer being freed was not allocated
4bp(10065,0x1e0f0db40) malloc: *** set a breakpoint in malloc_error_break to debug
Abort trap: 6
```

4 body problem plots (earth moon sun spaceship (?))





Zoomed in