Adaptive Step-Size Runge-Kutta for N-Body System

Kevin Juan (kj89)

October 12^{th} 2018

The Numerical Methods and Algorithms

The objective of this assignment was to implement a Runge-Kutta method that adjusts the step-size, h, automatically on a system of N first order ODE or an N^{th}-order ODE. The advantage of using an auto-step size Runge-Kutta method is the increase in efficiency. The efficiency increase comes from the fact that the step-szie can increase in regions where a smaller h is not necessary, thus cutting down superfluous calculations. For some problems, $\frac{h_{max}}{h_{min}} \sim \mathcal{O}(10-1000)$, meaning a wide range of h can be used for the entire calculations. The goal of the algorithm is to develop a means of estimating the error at each step to control both h and the error. This can be done by implementing a higher order Runge-Kutta with an embedded lower order solution as shown below:

$$\vec{y}_n = h\vec{y}(t_n) \tag{1}$$

$$\vec{k}_1 = h\vec{f}(t_n, \vec{y}_n) \tag{2}$$

$$\vec{k}_2 = h\vec{f}(t_n + c_2 h, \vec{y}_n + a_{21}\vec{k_1}) \tag{3}$$

$$\vec{k}_3 = h\vec{f}(t_n + c_3 h, \vec{y}_n + a_{31}\vec{k_1} + a_{32}\vec{k_2}) \tag{4}$$

$$\vec{k}_4 = h\vec{f}(t_n + c_4 h, \vec{y}_n + a_{41}\vec{k_1} + a_{42}\vec{k_2} + a_{43}\vec{k_3}) \tag{5}$$

$$\vec{k}_5 = h\vec{f}(t_n + c_5 h, \vec{y}_n + a_{51}\vec{k_1} + ... + a_{54}\vec{k_4}) \tag{6}$$

$$\vec{k}_6 = h\vec{f}(t_n + c_6 h, \vec{y}_n + a_{61}\vec{k_1} + ... + a_{65}\vec{k_5}) \tag{7}$$

$$\vec{y}_{n+1} = \vec{y}_n + b_1\vec{k}_1 + b_2\vec{k}_2 + ... + b_6\vec{k}_6 + \mathcal{O}(h^6) \tag{8}$$

$$\vec{y}^*_{n+1} = \vec{y}_n + b^*_1\vec{k}_1 + b^*_2\vec{k}_2 + ... + b^*_6\vec{k}_6 + \mathcal{O}(h^5) \tag{9}$$

The coefficients a_{ij}, b_j, b^*_j, and c_k have been previously tabulated. For the purpose of this assignment, the Dormand-Prince coefficients will be used, and are listed in §17.2 in Press et al[1]. When using these coefficients, Equation 9 must be modified as such:

$$\vec{y}^*_{n+1} = \vec{y}_n + b^*_1\vec{k}_1 + b^*_2\vec{k}_2 + ... + b^*_6\vec{k}_6 + b^*_7\vec{f}(t_n + h, \vec{y}_{n+1}) + \mathcal{O}(h^5) \tag{10}$$

Given a desired accuracy ϵ_{max}, we can estimate the optimal h, h_{opt}, according to the following steps. We first calculate the error in each step as shown below:

$$\epsilon_{n+1}^{(i)} = \vec{y}_{n+1}^{(i)} - \vec{y}_{n+1}^{(i)*} \tag{11}$$

We then need to define a scale to turn the error into a relative error. The scale can either be a user-defined constant, or calculated as follows:

$$\text{Scale}^{(i)} \sim |\vec{y}_n^{(i)}| + \left| h\frac{d\vec{y}^{(i)}}{dt} \right| + |\epsilon|_{min} \tag{12}$$

In this case, $\frac{d\vec{y}^{(i)}}{dt}$ is equivalent to $\vec{k}_1$. The maximum relative error, Δ_{max} is then determined by maximizing $\epsilon_{n+1}^{(i)}$ for $i = 1, 2, ..., N$ where N is the number of equations. If $\Delta_{max} \ll \epsilon_{max}$, $\vec{y}_{n+1}$ is saved, $t_{n+1} = t_n + h$, and h is increased according to the equation below:

$$h \leftarrow h_{opt} = h_{current} \left| \frac{\epsilon_{max}}{\Delta_{max}} \right|^{1/5} \tag{13}$$

If $\Delta_{max} \gg \epsilon_{max}$, we can set h to $\frac{h}{2}$ or $\frac{h}{5}$ and repeat the calculation with the new h. For this assignment, the latter was used to decrease h. To avoid an infinite loop, the decrease in h is aborted if $|h| < |h_{min}|$.

N-Body Interactions

This assignment looks at the 3-body interaction between 3 planetary objects: the Earth, moon, and a smaller third object denoted as moon-2. In 2-body problems, the position and velocity of the objects can be determined analytically, however, the addition of one or more bodies to the two body system necessitates a numerical solution. This is due to the fact that the interactions of all other objects on the object in question must be accounted for. This type of 3-body planetary system exhibits sensitivities to small changes in initial conditions as demonstrated in the next section. Due to this, an N-bodied system for $N > 2$ can generate a chaotic system.

For planetary interactions of $N > 2$, the following equations must be solved for:

$$\frac{d\vec{v}_i}{dt} = \frac{1}{m_i}\vec{F}_i = \sum_{j \neq i} m_j \frac{\vec{x}_j - \vec{x}_i}{|\vec{x}_j - \vec{x}_i|^3} \tag{14}$$

$$\frac{d\vec{x}_i}{dt} = \vec{v}_i \tag{15}$$

The subscripts i and j denote the two objects interacting with each other. For this assignment, the initial conditions in mks units on a Cartesian coordinate system were as follows.

Quantity	Earth	Moon	Moon-2
x	0.0	0.0	-4.97×10^8
y	0.0	3.84×10^8	0.0
v_x	-12.593	1019.0	965.0
v_y	0.0	0.0	820.0

Table 1: Initial conditions for 3-body interaction simulation.

A list of relevant constants for the objects in the simulation are listed as follows:

Quantity	Definition	Value
M_e	Mass of Earth	5.976×10^24 kg
M_m	Mass of the Moon	$0.0123 \times M_e$
M_{m2}	Mass of Moon-2	$0.2 \times M_m$
T_m	Orbital Period of the Moon	648 hours
G	Gravitational Constant	6.674×10^{-11} Nm^2/kg^2
R_e	Radius of Earth	6,378 km
R_m	Radius of the Moon	3,476 km
R_{m2}	Radius of Moon-2	$0.5 \times R_m$

Table 2: Relevant physical constants for the 3-body planetary simulation.

Implementation and Results

A general subroutine called `rkAdapt` was written using pointers to arrays and dynamic memory to implement the automatic step-size Runge-Kutta method for any system of ODEs. The use of dynamic memory could help the code run more efficiently and will be useful when working with large arrays especially with this system requiring 12 equations. The subroutine was first tested on the harmonic oscillator system with no damping or external driving force to ensure that it worked properly. The system of ODEs for the harmonic oscillator staring from a second order ODE is shown below:

$$\frac{d^2y}{dt^2} + \omega^2 y = 0 \tag{16}$$

$$\frac{dy_0}{dt} = y_1 \tag{17}$$

$$\frac{dy_1}{dt} = -\omega^2 y_0 \tag{18}$$

This particular system with initial conditions $y(t = 0) = 1$ and $y'(t = 0) = 0$ has a known solution:

$$y(t) = \cos(\omega t) \tag{19}$$

The `rkAdapt` subroutine was tested on this system for $\omega = 1.0$, which produces a normal cosine wave. The solution produced by the subroutine was compared to a built-in cosine function to confirm correctness.

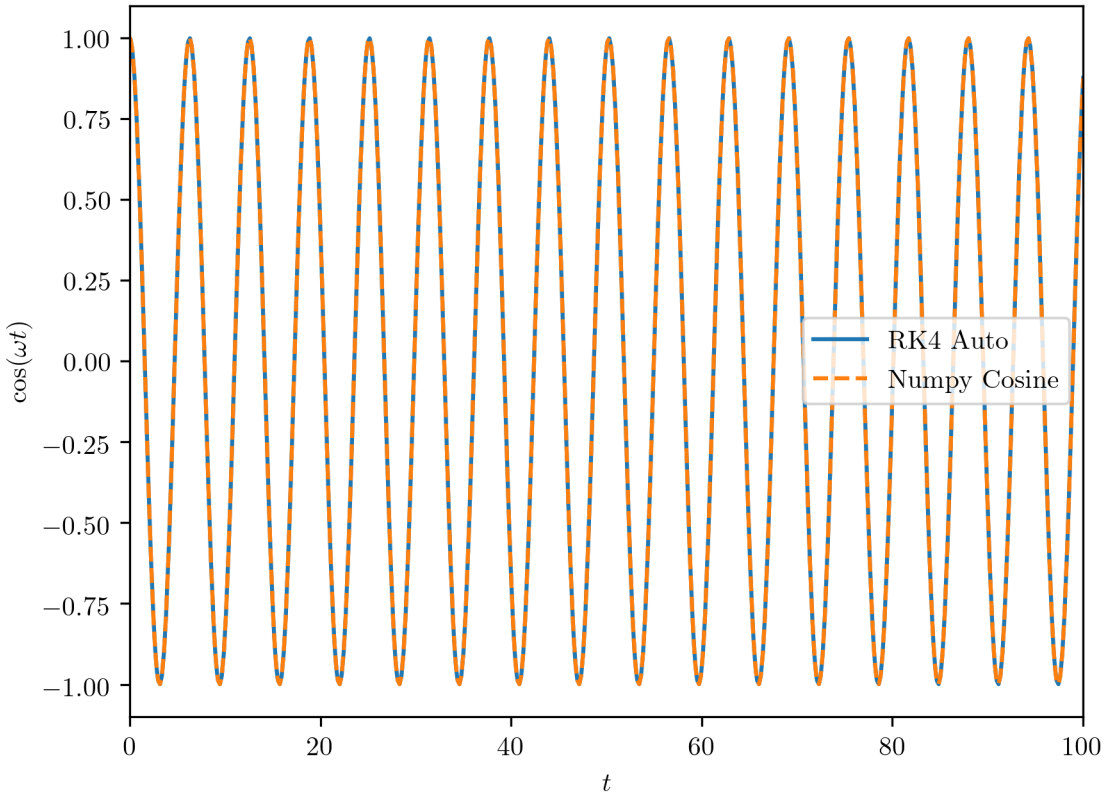

Figure 1: The cosine wave for the harmonic oscillator solution compared against Numpy's built-in cosine function, showing a high degree of accuracy.

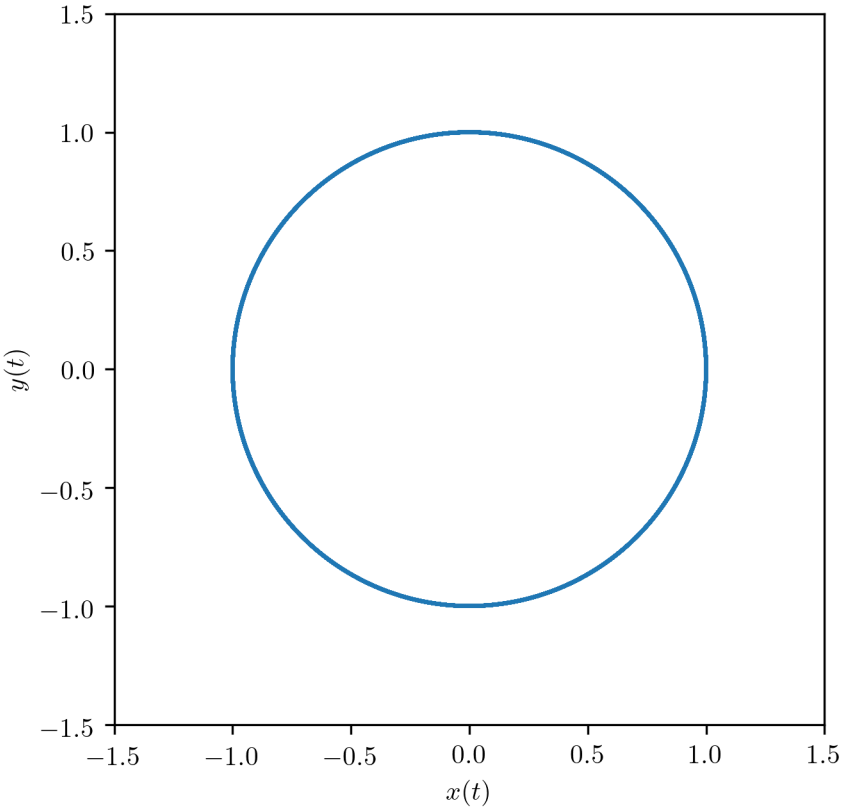

Figure 2: The phase space plot of the harmonic oscillator.

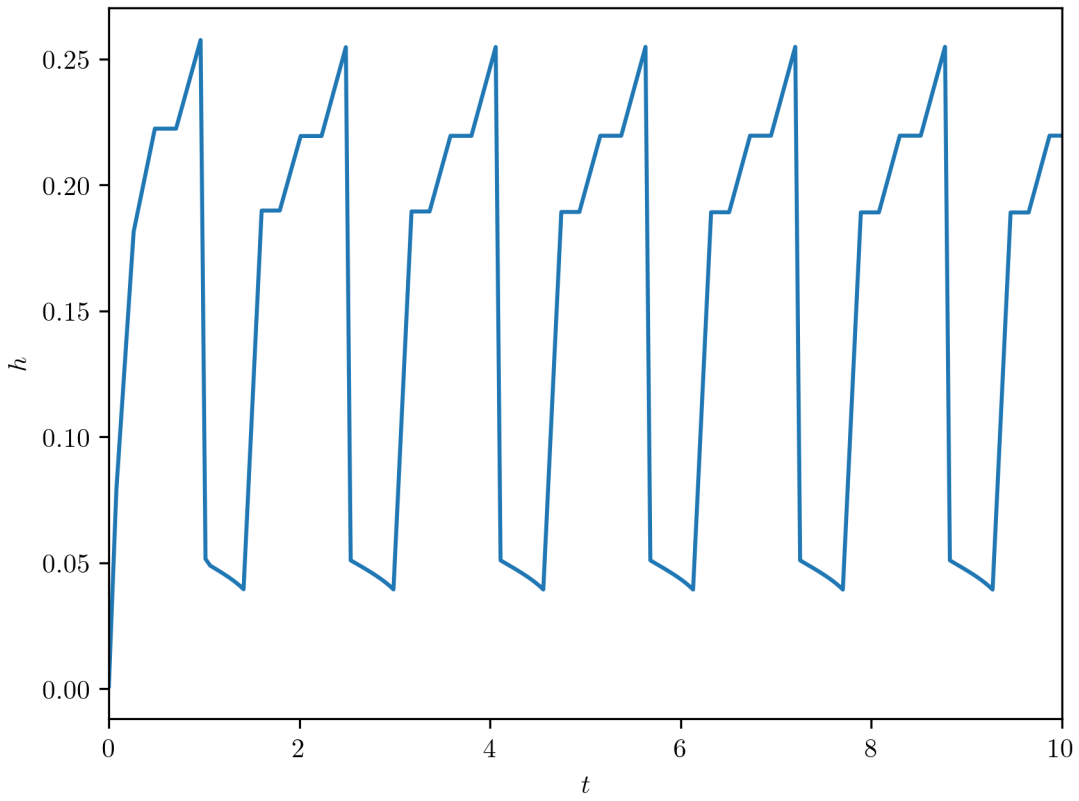

Figure 3: h vs. t for the harmonic oscillator.

As showing in Figure 3, h decreases as the subroutine reaches the peaks and valleys of the cosine wave while h increases in the linear portions of the wave. This should be expected since the peaks and valleys are rapid changes in concavity that require more refined calculations to capture correctly.

The auto step-size algorithm was also tested on a 2-body system in which only the Earth and moon were considered. This was done by making the moon-2 mass negligible compared to the other bodies.

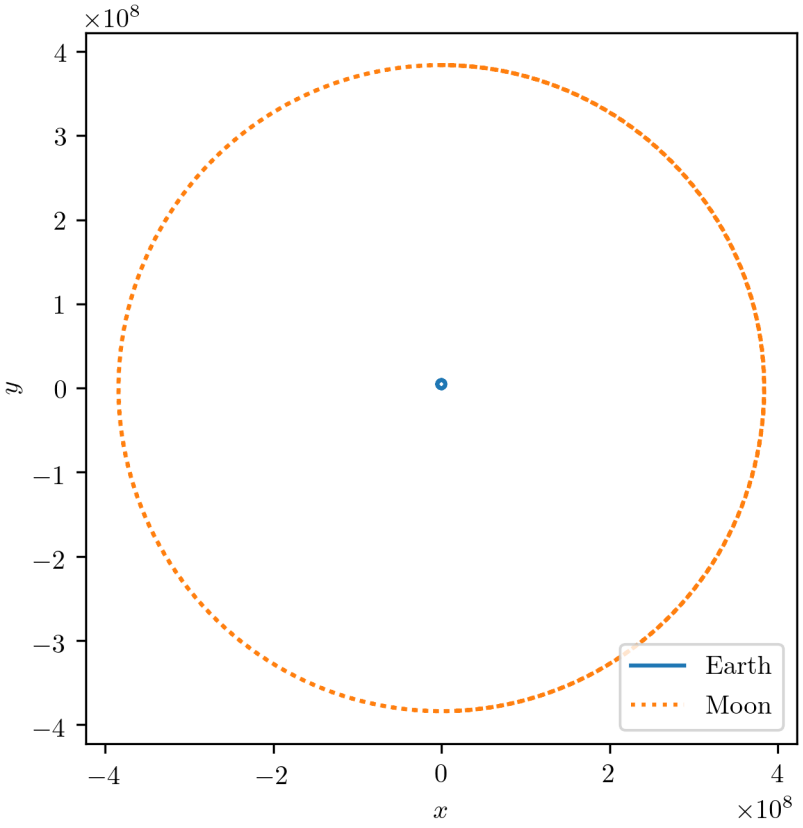

Figure 4: Orbital trajectories for the Earth and moon in a 2-body system.

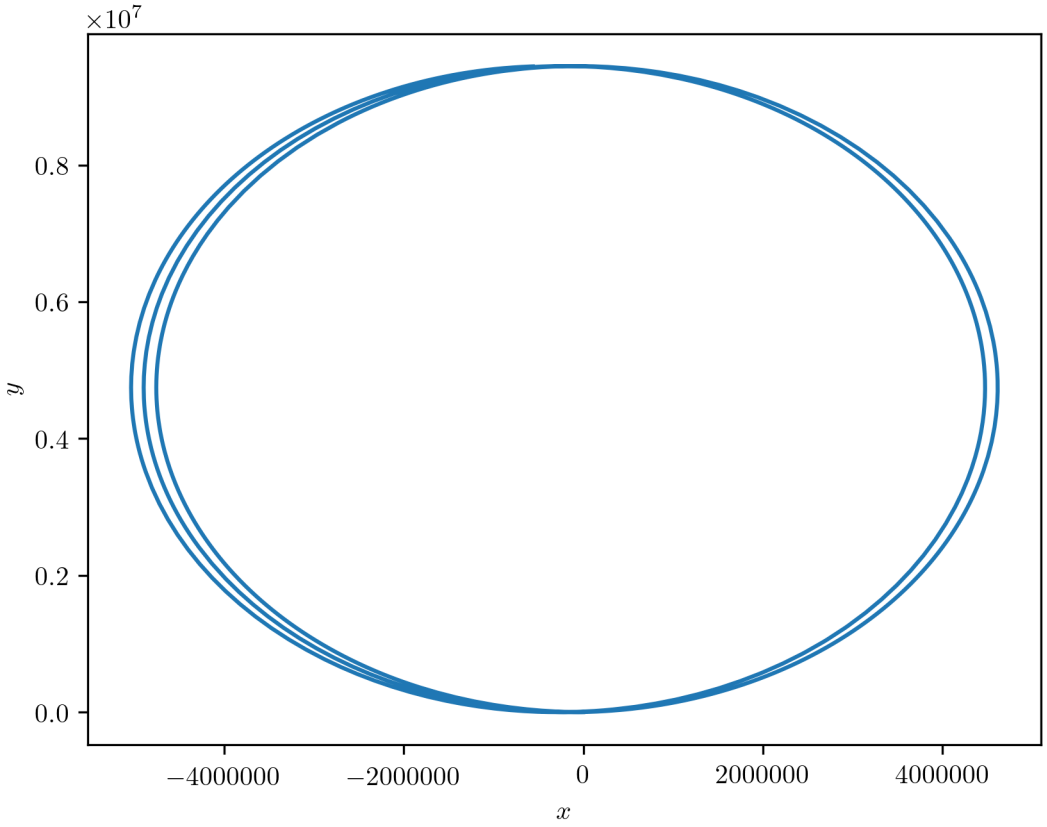

Figure 5: Orbital trajectory for the Earth in a 2-body system.

In the 2-body system, the Earth and moon have stable orbital trajectories as expected for this system. The program was then tested on the 3-body system with the initial conditions listed above for a time of 200 days. The plots below show the results of this simulation.

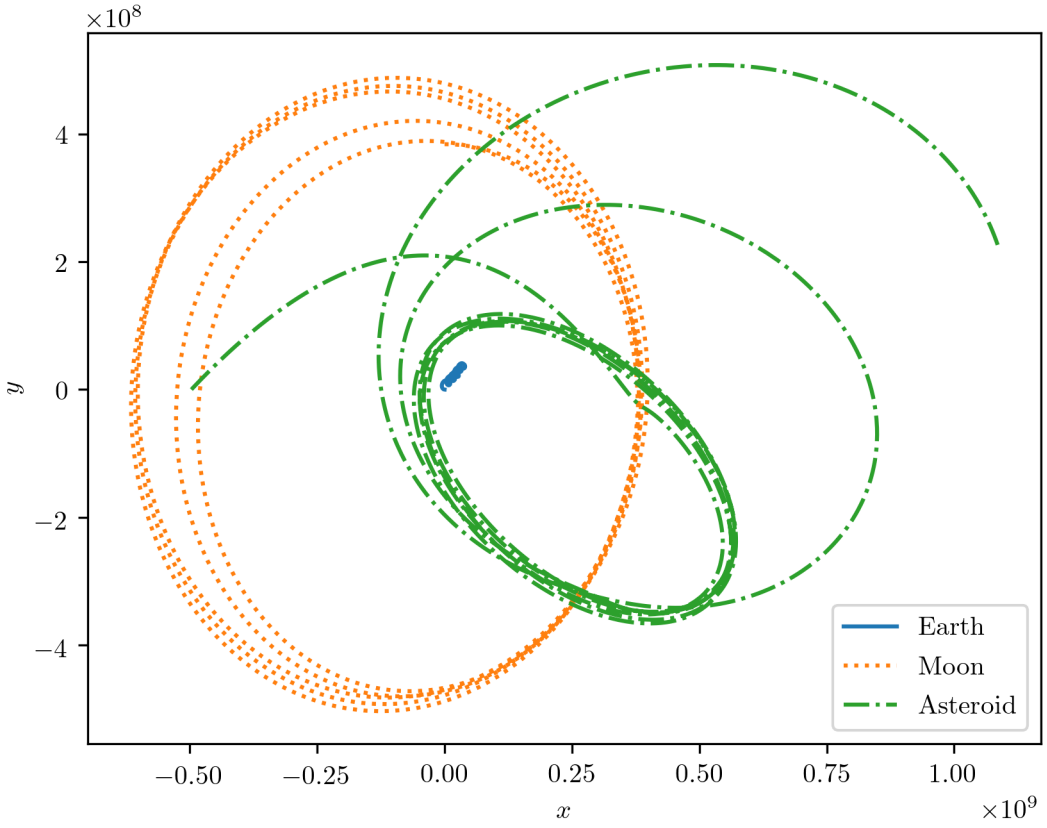

Figure 6: Orbital trajectory for the Earth, Moon, and Asteroid in a 3-body system.

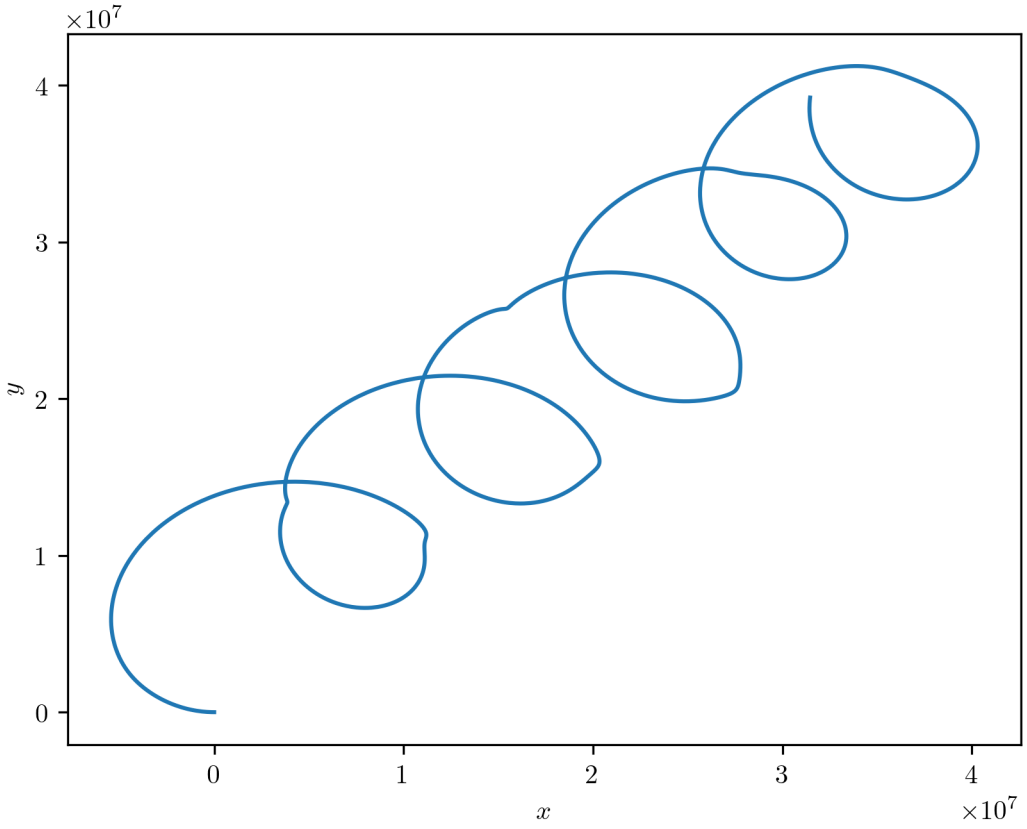

Figure 7: Orbital trajectory for Earth in the 3-body system.

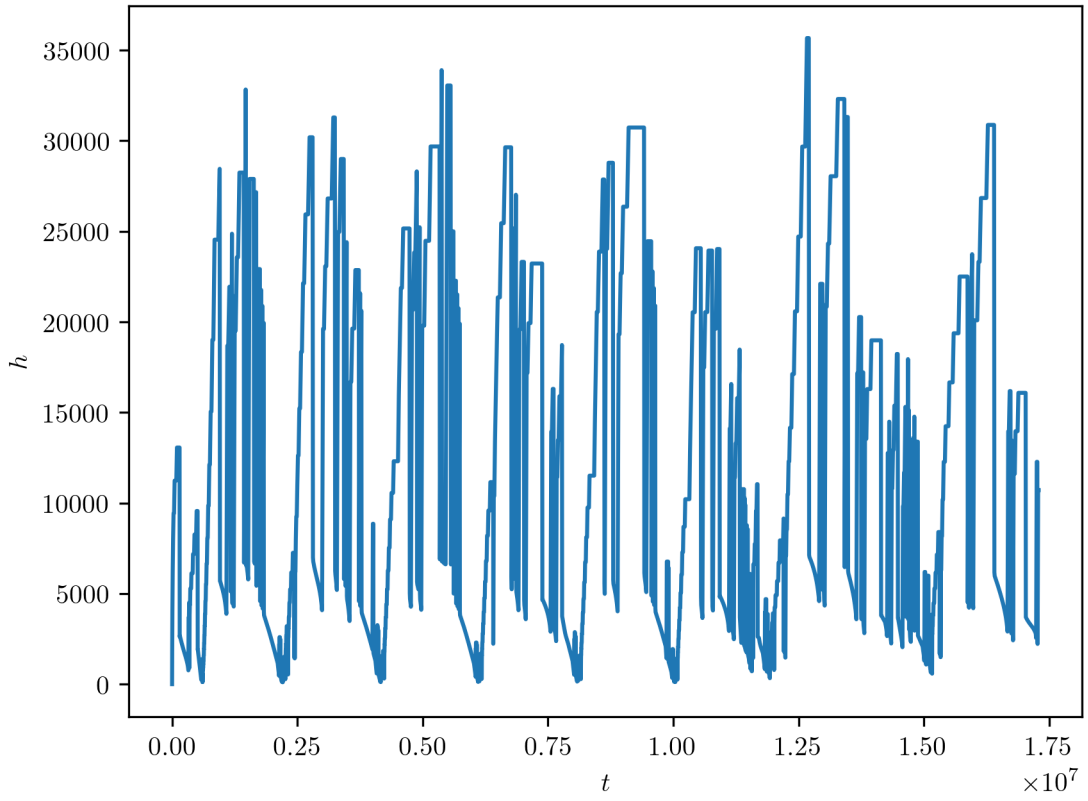

Figure 8: h vs. t for the 3-body system.

Figures 6 and 7 show that the addition of a third object causes unstable orbital trajectory for the Earth. The moon's orbital trajectory also shows a drastic change with the introduction of the third body. When the x-coordinate for Earth was altered to $x = 1000.0$, a relatively small length scale for the simulation under consideration, the following plot shows the change in trajectories:

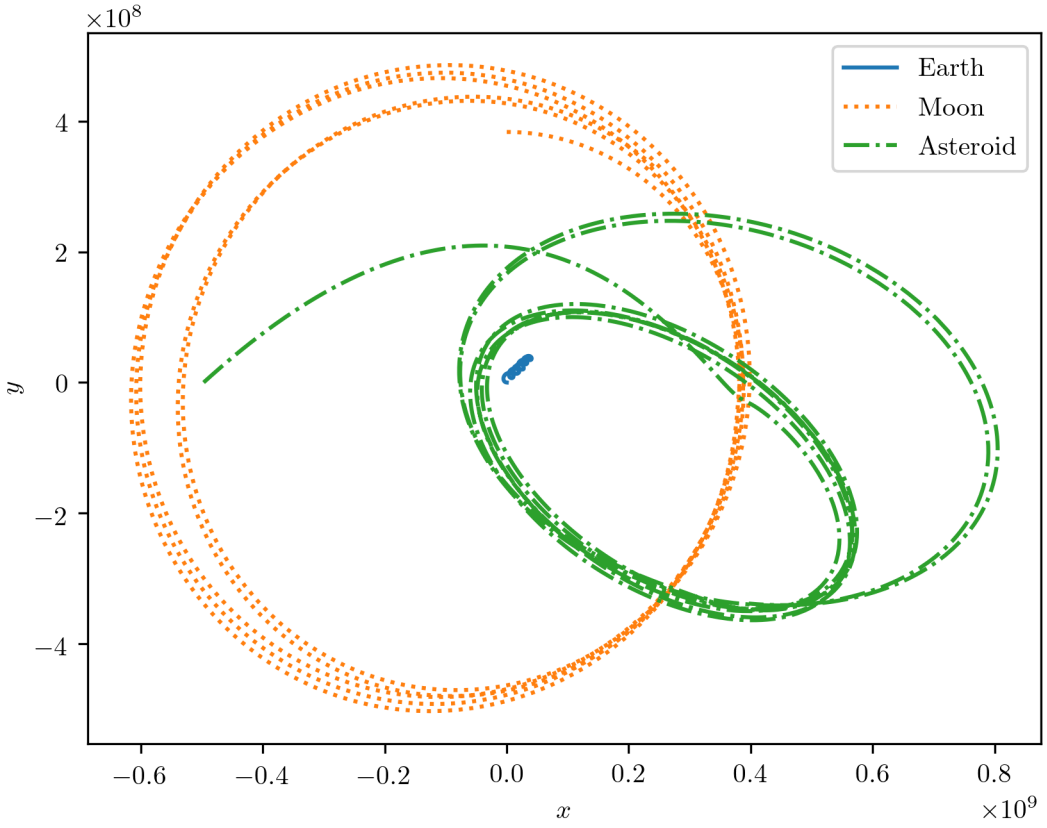

Figure 9: Orbital trajectory for the Earth, Moon, and Asteroid in a 3-body system for $x_e = 1000.0$.

This change in x_e produced a noticeable change in the moon and asteroid trajectories. The trajectory for Earth was nearly identical to the original trajectory. This drastic change in orbits for a relatively small change in initial condition shows that such a system can be chaotic in nature. Small changes in velocity or other coordinates yielded similar results, or potential collisions between bodies.

Source Code

```
/* AEP 4380 Homework #5

Test automatic step size ode solver
Runge-Kutta using automatic step sizing is tested on a harmonic oscillator
and three body problem

Run on a core i7 using clang 1000.11.45.2 on macOS Mojave
```

```
Kevin Juan 12 October 2018
*/

#include <cstdlib> // plain C
#include <cmath>   // use math package

#include <iostream> // stream IO
#include <fstream>  // stream file IO
#include <iomanip>  // to format the output

using namespace std;

void rkAdapt(double yold[], double ynew[], double h, int nEqns, void
frhs(double[], double, double[]), ofstream &fp, double tMin, double tMax,
double tol)
{
    int i;
    double *k1, *k2, *k3, *k4, *k5, *k6, *temp, *scale, *yerr, *ynewSt,
    *delta;
    const static double c2 = 0.2, c3 = 0.3, c4 = 0.8, c5 = 8.0 / 9.0,
    c6 = 1.0;
    const static double a21 = 0.2, a31 = 3.0 / 40.0, a32 = 9.0 / 40.0;
    const static double a41 = 44.0 / 45.0, a42 = -56.0 / 15.0,
    a43 = 32.0 / 9.0;
    const static double a51 = 19372.0 / 6561.0, a52 = -25360.0 / 2187.0,
    a53 = 64448.0 / 6561.0, a54 = -212.0 / 729.0;
    const static double a61 = 9017.0 / 3168.0, a62 = -355.0 / 33.0,
    a63 = 46732.0 / 5247.0, a64 = 49.0 / 176.0, a65 = -5103.0 / 18656.0;
    const static double b1 = 35.0 / 384.0, b2 = 0.0, b3 = 500.0 / 1113.0,
    b4 = 125.0 / 192.0, b5 = -2187.0 / 6784.0, b6 = 11.0 / 84.0;
    const static double bSt1 = 5179.0 / 57600.0, bSt2 = 0.0,
    bSt3 = 7571.0 / 16695.0, bSt4 = 393.0 / 640.0,
    bSt5 = -92097.0 / 339200.0, bSt6 = 187.0 / 2100.0, bSt7 = 1.0 / 40.0;
    double deltaMax, t = tMin;
    const static double smallVal = 1.0e-30;

    k1 = new double[11 * nEqns];
    k2 = k1 + nEqns;
    k3 = k2 + nEqns;
    k4 = k3 + nEqns;
    k5 = k4 + nEqns;
    k6 = k5 + nEqns;
    temp = k6 + nEqns;
    ynewSt = temp + nEqns;
```

```
scale = ynewSt + nEqns;
yerr = scale + nEqns;
delta = yerr + nEqns;

// output initial values to file
fp << setw(17) << t << setw(17) << h;
for (i = 0; i < nEqns; i++)
{
    fp << setw(17) << yold[i];
}
fp << endl;

while (t <= tMax)
{
    frhs(yold, t, k1);
    for (i = 0; i < nEqns; i++)
    {
        temp[i] = yold[i] + h * a21 * k1[i];
    }

    frhs(temp, t + c2 * h, k2);
    for (i = 0; i < nEqns; i++)
    {
        temp[i] = yold[i] + h * (a31 * k1[i] + a32 * k2[i]);
    }

    frhs(temp, t + c3 * h, k3);
    for (i = 0; i < nEqns; i++)
    {
        temp[i] = yold[i] + h * (a41 * k1[i] + a42 * k2[i] +
        a43 * k3[i]);
    }

    frhs(temp, t + c4 * h, k4);
    for (i = 0; i < nEqns; i++)
    {
        temp[i] = yold[i] + h * (a51 * k1[i] + a52 * k2[i] +
        a53 * k3[i] + a54 * k4[i]);
    }

    frhs(temp, t + c5 * h, k5);
    for (i = 0; i < nEqns; i++)
    {
        temp[i] = yold[i] + h * (a61 * k1[i] + a62 * k2[i] + a63 *
        k3[i] + a64 * k4[i] + a65 * k5[i]);
```

```
}

frhs(temp, t + c6 * h, k6);
for (i = 0; i < nEqns; i++)
{
    ynew[i] = yold[i] + h * (b1 * k1[i] + b2 * k2[i] + b3 *
    k3[i] + b4 * k4[i] + b5 * k5[i] + b6 * k6[i]);
}

frhs(ynew, t + h, temp);
for (i = 0; i < nEqns; i++)
{
    ynewSt[i] = yold[i] + h * (bSt1 * k1[i] + bSt2 * k2[i] +
    bSt3 * k3[i] + bSt4 * k4[i] + bSt5 * k5[i] + bSt6 * k6[i]
    + bSt7 * temp[i]);
}

deltaMax = -1000.0; // initialize an impossible deltaMax
for (i = 0; i < nEqns; i++)
{
    yerr[i] = abs(ynew[i] - ynewSt[i]);
    scale[i] = abs(ynew[i]) + abs(h * k1[i]) + smallVal;
    delta[i] = yerr[i] / scale[i];
    if (delta[i] > deltaMax)
    {
        deltaMax = delta[i];
    }
}

if (deltaMax < tol)
{
    t += h;
    fp << setw(17) << t << setw(17) << h;
    for (i = 0; i < nEqns; i++)
    {
        fp << setw(17) << ynew[i];
        yold[i] = ynew[i];
    }
    fp << endl;
    if (deltaMax < tol / 2.0)
    {
        h = h * pow(abs(tol / deltaMax), 1.0 / 5.0);
    }
}
else
```

```
        {
            h = h / 5.0;
        }
        cout << "t: " << t << endl;
    }
    delete[] k1;
    return;
}

void harmonicosc(double y[], double t, double f[])
// harmonic oscillator rhs
{
    f[0] = y[1];  // dy_0/dt
    f[1] = -y[0]; // dy_1/dt
    return;
}

void planets(double y[], double t, double f[])
// planetary motion rhs
{
    int N = 3, xo = 0, yo = N, vxo = 2 * N, vyo = 3 * N, i, j;
    double distCube, dx, dy, dist;
    const static double G = 6.674e-11;
    const static double mass[] = {5.976e24, 0.0123 * 5.976e24, 0.2 *
    0.0123 * 5.976e24}; // mass in kg
    const static double radii[] = {6378e3, 3476e3, 0.5 * 3476e3};
    // radii in meters

    for (i = 0; i < N; i++)
    {
        f[i + vxo] = 0.0; // initialize rhs for dv_x/dt
        f[i + vyo] = 0.0; // initialize rhs for dv_y/dt
        for (j = 0; j < N; j++)
        {
            if (i != j)
            {
                dx = y[j + xo] - y[i + xo];
                dy = y[j + yo] - y[i + yo];
                dist = sqrt(dx * dx + dy * dy);
                distCube = dist * dist * dist;
                if (dist <= radii[i] + radii[j])
                {
                    cout << "A collision has occurred" << endl;
                    exit(1);
                }
```

```
                f[i + vxo] += G * mass[j] * dx / distCube; // dv_x/dt
                f[i + vyo] += G * mass[j] * dy / distCube; // dv_y/dt
            }
        }
        f[i + xo] = y[i + vxo]; // dx_i/dt
        f[i + yo] = y[i + vyo]; // dy_i/dt
    }
    return;
}

int main()
{
    int nEqns = 12;
    double yold[nEqns], ynew[nEqns], h = 0.001;
    const static double convertDay = 60.0 * 60.0 * 24.0;
    // day to second conversion

    // // Runge-Kutta for harmonic oscillator
    // yold[0] = 1.0;
    // yold[1] = 0.0;

    // ofstream fp1;                  // output file using streams
    // fp1.open("harmonic_osc.dat"); // open new file for output
    // fp1.precision(9);             // select 9 digits

    // if (fp1.fail())
    // {
    //     // or fp.bad()
    //     cout << "cannot open file" << endl;
    //     return (EXIT_SUCCESS);
    // }

    // rkAdapt(yold, ynew, h, nEqns, harmonicosc, fp1, 0.0, 100.0, 10e-7);

    // fp1.close();

    // Runge-Kutta for 3-body motion
    yold[0] = 0.0;     // x_earth
    yold[1] = 0.0;     //x_moon
    yold[2] = -4.97e8; //x_moon-2
    yold[3] = 0.0;     // y_earth
    yold[4] = 3.84e8;  //y_moon
    yold[5] = 0.0;     // y_moon2
    yold[6] = -12.593; //vx_eartg
    yold[7] = 1019.0;  //vx_moon
```

```
    yold[8] = 965.0;   //vx_moon-2
    yold[9] = 0.0;     //vy_earth
    yold[10] = 0.0;    //vy_moon
    yold[11] = 820.0;  //vy_moon-2

    ofstream fp2;
    fp2.open("3_body_planets.dat");
    fp2.precision(9);

    if (fp2.fail())
    {
        cout << "cannot open file" << endl;
        return (EXIT_SUCCESS);
    }

    rkAdapt(yold, ynew, h, nEqns, planets, fp2, 0.0, 200.0 * convertDay,
    10e-10);

    fp2.close();
}
```

References

[1] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery, editors. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK ; New York, 3rd ed edition, 2007. OCLC: ocn123285342.