AEP 4380 HW#5: N-Body Problems

Gianfranco Grillo

October 14, 2016

1 Background

1.1 Numerical Solving of ODEs Using Runge-Kutta-Fehlberg

The Runge-Kutta-Fehlberg method for numerically solving a system of coupled ODEs is an extension of the more basic 4th/5th order regular Runge-Kutta algorithms that has the added advantage of being able to automatically adjust the step size h to be used by estimating the error at each iteration.

Given a system of coupled ODEs of the form

$$\frac{d\vec{x}}{dt} = \vec{f}(t,\, \vec{x}(t)) \tag{1}$$

With initial conditions

$$\vec{x}(t_0) = \vec{x_0} \tag{2}$$

Where t is an independent variable, the algorithm calculates six k parameters in order to progressively reconstruct the solutions of the ODE system, and use the difference between a high order solution and a low order solution to estimate the error resulting from the use of a particular h. These parameters are given by

$$\begin{aligned}
\vec{k_1} &= h\vec{f}(t_n, \vec{x_n}) && (3)\\
\vec{k_2} &= h\vec{f}(t_n + c_2 h,\, \vec{x_n} + a_{21}\vec{k_1}) && (4)\\
\vec{k_3} &= h\vec{f}(t_n + c_3 h,\, \vec{x_n} + a_{31}\vec{k_1} + a_{32}\vec{k_2}) && (5)\\
\vec{k_4} &= h\vec{f}(t_n + c_4 h,\, \vec{x_n} + a_{41}\vec{k_1} + a_{42}\vec{k_2} + a_{43}\vec{k_3}) && (6)\\
\vec{k_5} &= h\vec{f}(t_n + c_5 h,\, \vec{x_n} + a_{51}\vec{k_1} + a_{52}\vec{k_2} + a_{53}\vec{k_3} + a_{54}\vec{k_4}) && (7)\\
\vec{k_6} &= h\vec{f}(t_n + c_6 h,\, \vec{x_n} + a_{61}\vec{k_1} + a_{62}\vec{k_2} + a_{63}\vec{k_3} + a_{64}\vec{k_4} + a_{65}\vec{k_5}) && (8)
\end{aligned}$$

Where the c's and a's are coefficients known as the Dormund-Prince coefficients. The higher order solution is then given by

$$\vec{x_{n+1}} = \vec{x_n} + b_1\vec{k_1} + b_2\vec{k_2} + b_3\vec{k_3} + b_4\vec{k_4} + b_5\vec{k_5} + b_6\vec{k_6} + \mathcal{O}(h^6) \tag{9}$$

Whereas the lower order order solution is

$$\vec{x'_{n+1}} = \vec{x_n} + b'_1\vec{k_1} + b'_2\vec{k_2} + b'_3\vec{k_3} + b'_4\vec{k_4} + b'_5\vec{k_5} + b'_6\vec{k_6} + b'_7 h\vec{f}(t_n + h,\, \vec{x_{n+1}}) + \mathcal{O}(h^5) \tag{10}$$

Where b_n and b'_n are also Dormund-Prince coefficients. The error in the current step is then given by

$$\epsilon^{(i)}_{n+1} = \left| x^{(i)}_{n+1} - x'^{(i)}_{n+1} \right| \tag{11}$$

Where $i = 1, 2, 3, ..., N$. The error is then scaled in order to adjust for the different magnitudes and units of the variables using a scale given by

$$scale^{(i)} = \left|x_n^{(i)}\right| + \left|hf^{(i)}(t_n, y_n^{(i)}(t))\right| \tag{12}$$

This is used to find the maximum (relative) error Δ_{max} in the series, via

$$\Delta_{max} = max\left[\frac{\epsilon_{n+1}^{(i)}}{scale^{(i)}}\right] \tag{13}$$

Given a predetermined maximum allowed error ϵ_{max}, the algorithm changes the step size h in two cases. When $\Delta_{max} \ll \epsilon_{max}$, h is increased via

$$h_{new} = h_{old}\left(\frac{\epsilon_{max}}{\Delta_{max}}\right)^{1/5} \tag{14}$$

And when $\Delta_{max} > \epsilon_{max}$, h is reduced by an arbitrary factor n, giving

$$h_{new} = \frac{h_{old}}{n} \tag{15}$$

In the last case, the current iteration step is repeated. The final $\vec{x_{n+1}}$ corresponds to the next value of the solution.

1.2 N-Body Gravitational Problem

Isaac Newton famously claimed that the stable orbits of the planets in our solar system were very hard to explain without assuming that providence intervenes now and then in order to preserve stability; that the gravitational interactions between the planets would necessarily destabilize their orbits given a sufficiently long period of time. He had no way of knowing this, however, since he didn't know how to analytically solve the N-body problem involving all of the planetary interactions. French mathematical physicist Pierre-Simon Laplace would later show that it was indeed possible for the solar system to be stable without requiring any kind of divine intervention, but the problem is conclusively regarded as analytically unsolvable, and so we have to rely on approximations in order to describe the motions of the planets over time. Each planet's motion is governed by the following set of coupled ODEs (assuming all objects are in the same plane):

$$\frac{d\vec{v_i}}{dt} = \frac{\vec{F_i}}{m_i} = G\sum m_j \frac{\vec{x_j} - \vec{x_i}}{\|\vec{x_j} - \vec{x_i}\|^3} \tag{16}$$

$$\frac{d\vec{x_i}}{dt} = \vec{v_i} \tag{17}$$

Given the initial positions and velocities, we can use Runge-Kutta-Fehlberg in order to plot the trajectories of the planets over a certain period of time. The goal of this homework is to plot the trajectories of the Earth, Mars, Jupiter, and the Sun over a period of 15 years given two different sets of initial conditions.

2 Results

2.1 Task 1

The resulting trajectory plot for the first set of initial conditions is shown in Figure 1. This was obtained by running hw5a.cpp, which uses Runge-Kutta-Fehlberg to output the coordinates of each planet over a period of 15 years to a file. Figure 2 shows an amplification of the Sun's motion for the same setup.

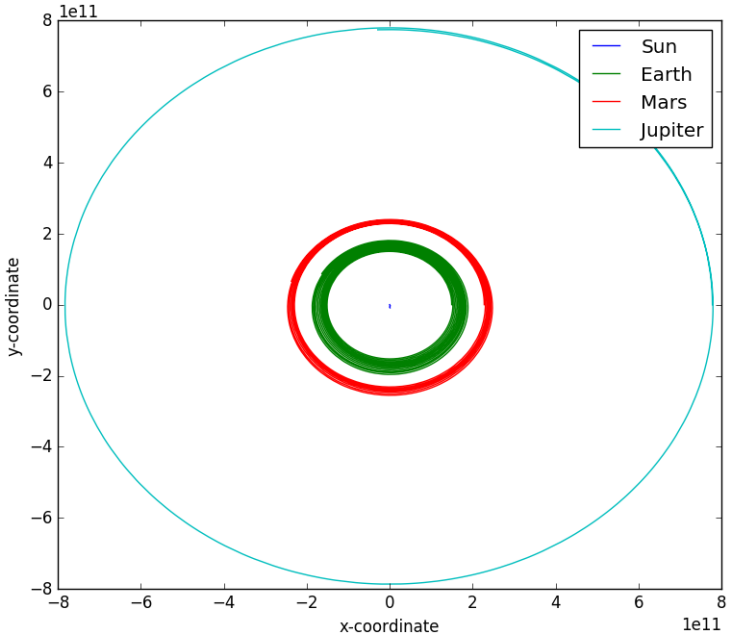

Figure 1: Solar system trajectories over 15 year period for first set of initial conditions

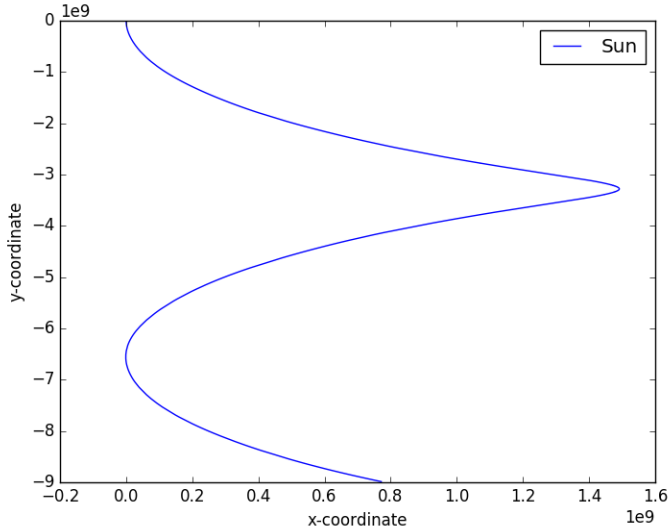

Figure 2: Sun's motion for first set of initial conditions.

2.2 Task 2

Figure 3 shows the resulting trajectory for the second set of initial conditions, which is different from the first one in that the velocity of Jupiter has been increased by a factor of $\sqrt{4.7}$ and its initial x-coordinate has been increased by a factor of 4.7. Figure 4 is the equivalent of Figure 2 for this set of initial conditions.

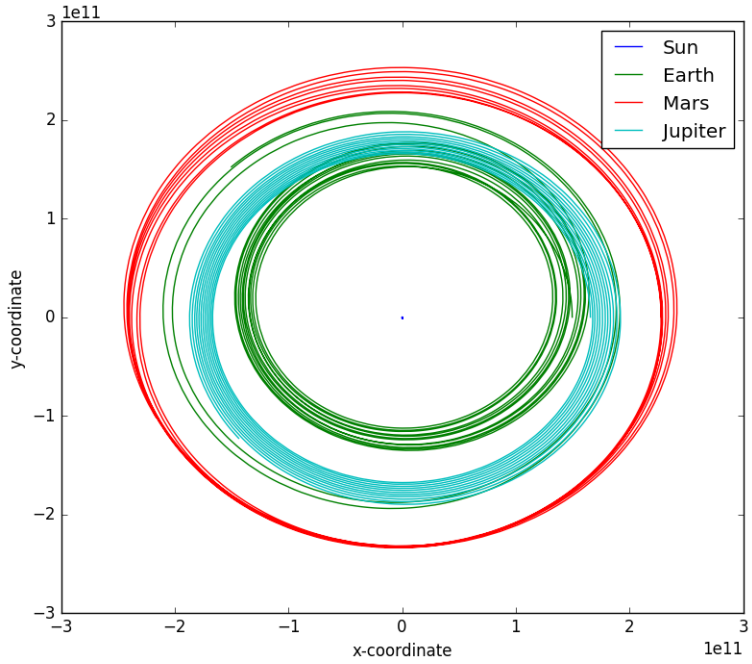

Figure 3: Solar system trajectories over 15 year period for second set of initial conditions.

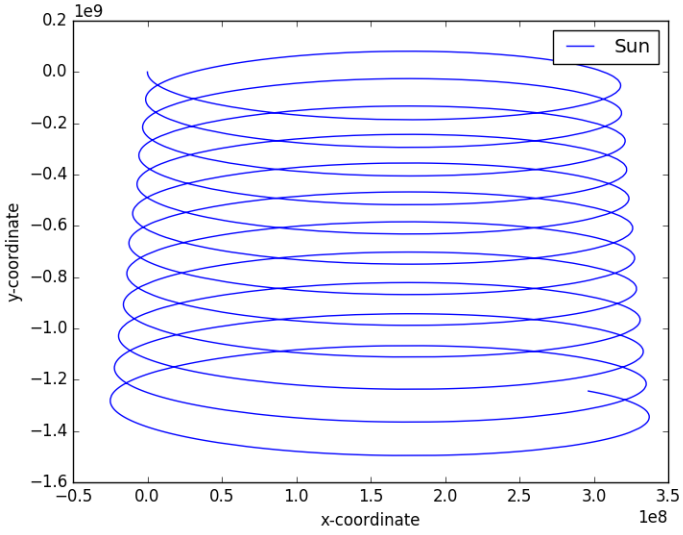

Figure 4: Sun's motion for second set of initial conditions.

3 Analysis

What the results show are that the first set of initial conditions result in a stable system with almost perfectly circular orbits, whereas the second set of initial conditions produce an unstable system with spiral orbits and potentially chaotic motion, in particular in the case of the motions of Mars and the Earth, which have smaller masses and therefore are easier to be upset by gravitational interactions. For the case of the Sun, we can see that it moves much less in the first case than in the second case, basically due to the fact that in the first case Jupiter is farther away, and as such exerts less gravitational force and also takes longer to complete a full orbit, while in the second case Jupiter's period decreases and its gravitational pull on the Sun increases, as it is much closer to it. Because the Earth and Mars are not big enough to significantly upset the Sun's motion, we see that the way the Sun moves is much more sensitive on Jupiter's gravitational attraction than on Earth and Mars's.

4 Source code

4.1 hw5a.cpp

```
/* AEP 4380 HW#5a
   Autostep size RK and Planetary Motion
   Run on core i7 with g++ 5.4.0 (Ubuntu)
   Gianfranco Grillo 10/14/2016
*/

#include <cstdlib> // plain C
#include <cmath>

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

using namespace std;

int main() {
        int istep, n = 16, i;
        double initvar[n] = {149.598e9, 228.0e9, 778.29e9, 0.0, 0.0, 0.0, 0.0, 0.0,
             0.0, 0.0,
        0.0, 0.0, 2.9786e4, 2.4127e4, 1.30588e4, -3.0e1};
        double finalvar[16], t0 = 0.0, hold, h, nsecinday, totalsec, nsecinyear;
        void autosteprk4(double[], double[], int, double&, double, void(double[],
            double, double[], double));
        void nbody(double[], double, double[], double);
        ofstream fp;
    fp.open( "solarsystem1.dat" ); // open new file for output
        if( fp.fail() ) { // in case of error
        cout << "cannot open file" << endl;
        return( EXIT_SUCCESS );
        }
        nsecinyear = 365.0*24.0*60.0*60.0;
        nsecinday = 3600.0*24.0;
        h = nsecinday; // initial h
        totalsec = 15.0*nsecinyear;
```

```
        fp << setw(15) << "t (years)" << setw(15) << "xearth" << setw(15) << "
            yearth" << setw(15) << "xmars" << setw(15)
        << "ymars" << setw(15) << "xjupiter" << setw(15) << "yjupiter" << setw(15)
            << "xsun" << setw(15) << "ysun" << endl;
        while (t0 < totalsec) {
                hold = h;
                autosteprk4(initvar, finalvar, n, h, t0, nbody);
                if (h > hold || h == hold) { // means error is low enough, result
                    is correct
                        for (i = 0; i < n; i++) { initvar[i] = finalvar[i]; }
                        fp <<  setw(15) <<  t0/nsecinyear << setw(15) << finalvar
                            [0] << setw(15) << finalvar[4] << setw(15)
                        << finalvar[1] << setw(15) << finalvar[5] << setw(15) <<
                            finalvar[2] << setw(15) << finalvar[6]
                        << setw(15) << finalvar[3] << setw(15) << finalvar[7] <<
                            endl;
                        t0 = t0 + hold;
                }
    }
    fp.close();
    return( EXIT_SUCCESS );
}

void autosteprk4(double initvar[], double highfinalvar[], int n, double& h,
                                double t0, void functions(double[], double, double
                                    [], double)) {
        int i;
        double scale[n], error[n], maxerror = 0.00000005, lowfinalvar[n], deltamax;
        double maximum(double[], int);
        double *k1, *k2, *k3, *k4, *k5, *k6, *k7, *scalek, *temp;
        double c2 = 0.2, c3 = 0.3, c4 = 0.8, c5 = 8.0/9.0, c6 = 1.0, c7 = 1.0;
        double a21 = 0.2, a31 = 3.0/40.0, a32 = 9.0/40.0, a41 = 44.0/45.0, a42 =
            -56.0/15.0, a43 = 32.0/9.0;
        double a51 = 19372.0/6561.0, a52 = 25360.0/2187.0, a53 = 64448.0/6561.0,
            a54 = -212.0/729.0;
        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;
        double b1 = 35.0/384.0, b3 = 500.0/1113.0, b4 = 125.0/192.0, b5 =
            -2187.0/6784.0, b6 = 11.0/84.0;
        double d1 = 5179.0/57600.0, d3 = 7571.0/16695.0, d4 = 393.0/640.0, d5 =
            -92097.0/339200.0, d6 = 187.0/2100.0, d7 = 1.0/40.0;
        k1 = new double[10*n];
        if (NULL == k1) {
                cout << "can't allocate arrays in rk4" << endl;
        return;
        }
        k2 = k1 + n;
        k3 = k2 + n;
        k4 = k3 + n;
        k5 = k4 + n;
        k6 = k5 + n;
        k7 = k6 + n;
```

```
temp = k7 + n;
scalek = temp + n; // allocate dynamic memory to variables
functions(initvar, t0, k1, h); // generate k1
for (i = 0; i < n; i++) {
        temp[i] = initvar[i] + a21*k1[i];
}
functions(temp, t0 + c2*h, k2, h);
for (i = 0; i < n; i++) {
        temp[i] = initvar[i] + a31*k1[i] + a32*k2[i];
}
functions(temp, t0 + c3*h, k3, h);
for (i = 0; i < n; i++) {
        temp[i] = initvar[i] + a41*k1[i] + a42*k2[i] + a43*k3[i];
}
functions(temp, t0 + c4*h, k4, h);
for (i = 0; i < n; i++) {
        temp[i] = initvar[i] + a51*k1[i] + a52*k2[i] + a53*k3[i] + a54*k4[i
            ];
}
functions(temp, t0 + c5*h, k5, h);
for (i = 0; i < n; i++) {
        temp[i] = initvar[i] + a61*k1[i] + a62*k2[i] + a63*k3[i] + a64*k4[i
            ] + a65*k5[i];
}
functions(temp, t0 + c6*h, k6, h);
for (i = 0; i < n; i++) {
        highfinalvar[i] = initvar[i] + b1*k1[i] + b3*k3[i] + b4*k4[i] + b5*
            k5[i] + b6*k6[i];
}
functions(highfinalvar, t0 + h, k7, h);
for (i = 0; i < n; i++) {
        lowfinalvar[i] = initvar[i] + d1*k1[i] + d3*k3[i] + d4*k4[i] + d5*
            k5[i] + d6*k6[i] + d7*k7[i];
}
functions(initvar, t0 + h, scalek, h); // generate scale
for (i = 0; i < n; i++) {
        scale[i] = abs(initvar[i]) + abs(scalek[i]) + 0.01;
}
for (i = 0; i < n; i++) {
        error[i] = abs((highfinalvar[i] - lowfinalvar[i])/scale[i]);
}
deltamax = maximum(error, n);
if (5*deltamax < maxerror) {
        h = h*pow(maxerror/deltamax, 0.2); // if error is too small,
            increase h
        delete k1;
        return;
}
else if (deltamax > maxerror && h > 0.0000000000001) { // if error is too
    big, decrease h
        h = h/3.0;
        delete k1;
```

```
                return;
        }
        else {
                delete k1;
                return;
        }
}

void nbody(double var[], double t, double k[16], double h) {
        int nobj = 4, i, n, vxloc, vyloc;
        double masses[nobj] = {5.9742e24, 0.64191e24, 1898.8e24, 1.9891e30};
        double G = 6.6726e-11, xsingleacc, xdistance, ysingleacc, ydistance,
            totaldist, totaldistc;
        for (i = 0; i < nobj; i++) {
                vxloc = i+2*nobj;
                vyloc = i+3*nobj;
                k[i] = h*var[i+2*nobj];
                k[i+nobj] = h*var[i+3*nobj];
                k[vxloc] = 0;
                k[vyloc] = 0;
                for(n = 0; n < nobj; n++) {
                        if (n != i) {
                                xdistance = var[n] - var[i];
                                ydistance = var[n+nobj] - var[i+nobj];
                                totaldist = sqrt(xdistance*xdistance+ydistance*
                                    ydistance);
                                totaldistc = totaldist*totaldist*totaldist;
                                xsingleacc = h*G*masses[n]*xdistance/totaldistc;
                                k[vxloc] = k[vxloc] + xsingleacc;
                                ysingleacc = h*G*masses[n]*ydistance/totaldistc;
                                k[vyloc] = k[vyloc] + ysingleacc;
                        }
                }
        }
        return;
}

double maximum(double array[], int n) {
        double max;
        int i;
        max = array[0];
        for (i = 0; i < n; i++) {
                if (array[i] > max) {
                        max = array[i];
                }
        }
        return max;
}
```

4.2 hw5b.cpp

Identical to hw4a.cpp but main() is

```
int main() {
        int istep, n = 16, i;
        double initvar[n] = {149.598e9, 228.0e9, 778.29e9/4.7, 0.0, 0.0, 0.0, 0.0,
            0.0, 0.0, 0.0,
        0.0, 0.0, 2.9786e4, 2.4127e4, 1.30588e4*sqrt(4.7), -3.0e1};
        double finalvar[16], t0 = 0.0, hold, h, nsecinday, totalsec, nsecinyear;
        void autosteprk4(double[], double[], int, double&, double, void(double[],
            double, double[], double));
        void nbody(double[], double, double[], double);
        ofstream fp;
    fp.open( "solarsystem2.dat" ); // open new file for output
        if( fp.fail() ) { // in case of error
        cout << "cannot open file" << endl;
        return( EXIT_SUCCESS );
        }
        nsecinyear = 365.0*24.0*60.0*60.0;
        nsecinday = 3600.0*24.0;
        h = nsecinday; // initial h
        totalsec = 15.0*nsecinyear;
        fp << setw(15) << "t (years)" << setw(15) << "xearth" << setw(15) << "
            yearth" << setw(15) << "xmars" << setw(15)
       << "ymars" << setw(15) << "xjupiter" << setw(15) << "yjupiter" << setw(15)
           << "xsun" << setw(15) << "ysun" << endl;
        while (t0 < totalsec) {
                hold = h;
                autosteprk4(initvar, finalvar, n, h, t0, nbody);
                if (h > hold || h == hold) { // means error is low enough, result
                    is correct
                        for (i = 0; i < n; i++) { initvar[i] = finalvar[i]; }
                        fp <<  setw(15) <<  t0/nsecinyear << setw(15) << finalvar
                            [0] << setw(15) << finalvar[4] << setw(15)
                        << finalvar[1] << setw(15) << finalvar[5] << setw(15) <<
                            finalvar[2] << setw(15) << finalvar[6]
                        << setw(15) << finalvar[3] << setw(15) << finalvar[7] <<
                            endl;
                        t0 = t0 + hold;
                }
    }
    fp.close();
    return( EXIT_SUCCESS );
}
```

References

[1] Gillespie, C. C. *Pierre Simon Laplace 1749–1827: A Life in Exact Science*, Princeton: Princeton University Press (1997)

[2] Newton, I. *Opticks (2nd Edition)* (1706), quoted in H. G. Alexander (ed): *The Leibniz-Clarke correspondence*, University of Manchester Press (1957).