Application of 4^{th}-Order Runge-Kutta Method to Chaotic Systems

Kevin Juan (kj89)

October 3^{rd} 2018

The Numerical Methods and Algorithms

The objective of this assignment was to implement the 4^{th}-order Runge-Kutta method for solving a system of N-coupled 1^{st}-order differential equations or an N^{th}-order differential equation that can be turned into a system of 1^{st}-order differential equations. Compared to the simpler Euler's Method, the Runge-Kutta method provides a more accurate solution on the orders of $\mathcal{O}(h^5)$ locally and $\mathcal{O}(h^4)$ globally, where h is the step size. We can define an initial value problem where the vectors $\vec{f}$ and $\vec{y}$ are of length N as shown below:

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

$$\vec{y}(t = t_0) = \vec{y}_0 = \text{Constant} \tag{2}$$

The 4^{th}-order Runge-Kutta method performs 4 evaluations of $\vec{f}$: one at the starting point t, two at middle of $t + \frac{h}{2}$, and one at $t + h$. These are stored into the vectors $\vec{k}_1$, $\vec{k}_2$, $\vec{k}_3$, and $\vec{k}_4$, all of which have length N. As shown below, the vector $\vec{k}_j$ must be evaluated before the vector $\vec{k}_{j+1}$.

$$\vec{y}_n = \vec{y}(t_0 + nh) \tag{3}$$

$$\vec{k}_1 = \vec{f}(t_n, \vec{y}_n) \tag{4}$$

$$\vec{k}_2 = \vec{f}\left(t_n + \frac{h}{2}, \vec{y}_n + \frac{1}{2}\vec{k}_1\right) \tag{5}$$

$$\vec{k}_3 = \vec{f}\left(t_n + \frac{h}{2}, \vec{y}_n + \frac{1}{2}\vec{k}_2\right) \tag{6}$$

$$\vec{k}_4 = \vec{f}(t_n + h, \vec{y}_n + \vec{k}_3) \tag{7}$$

$$\vec{y}_{n+1} = \vec{y}_n + \frac{h}{6}(\vec{k}_1 + 2\vec{k}_2 + 2\vec{k}_3 + \vec{k}_4) + \mathcal{O}(h^5) \tag{8}$$

All 4 evaluations of $\vec{f}$ are summed up, putting greater weight on the midpoints, then multiplied by the step size and added to the prior function evaluations stored in the vector $\vec{y}_n$ to get the new function evaluations stored in vector $\vec{y}_{n+1}$. Adding weighted values of different derivative evaluations leads to a more accurate answer than Euler's method, which relies on only one derivative evaluation at $t + h$ to get a new function value. Due to its accuracy, the 4^{th}-order Runge-Kutta method has become a standard method for many ODE solvers. Despite this, the 4^{th}-order Runge-Kutta method is not always the most efficient way of solving ODEs. More detail on the Runge-Kutta method can be read about in §17.1 in Press et al[1] and §9.5.2 in Landau et al[2].

Chaotic Systems

A chaotic system is one that is bounded, but shows no obvious pattern or periodicity. At a bounded steady state, the behavior is not at an equilibrium point, not periodic, and not semi-periodic. These systems are deterministic and exhibit random-like behavior, and require a system of $N \geq 2$ that has some non-linearity. Chaotic systems are highly sensitive to initial conditions, and even the smallest perturbation could cause an arbitrarily large change much later in time. Because of this, chaotic systems are hard to predict. Some common examples of chaotic systems include weather, traffic, and the double-rod pendulum. Chaos can be read about in §12.4 in Landau et al[2].

This particular assignment examines the Jaynes-Cummings model for a light wave in a system of two level atoms. The derivation and description of the model for the set of equations listed below can be found in Ackerhalt et al[3].

$$\frac{dx}{dt} = -y \tag{9}$$

$$\frac{dy}{dt} = x + ez \tag{10}$$

$$\frac{dz}{dt} = -ey \tag{11}$$

$$\frac{d^2e}{dt^2} + \mu^2 e = \beta \frac{dy}{dt} \tag{12}$$

For applicability to the 4^{th}-order Runge-Kutta method, Equation 12 was broken into two first order differential equations as shown:

$$\frac{de_0}{dt} = e_1 \tag{13}$$

$$\frac{de_1}{dt} = \beta \frac{dy}{dt} - \mu^2 e_0 \tag{14}$$

The functions $x(t)$, $y(t)$, and $z(t)$ are related to the quantum mechanical amplitudes of each level, including the complex conjugates, while $e(t)$ relates to overall field strength.

The wavefunction for the atom can be written as a superposition of two states ψ_1 and ψ_2, giving the total wavefunction as such:

$$\psi(t) = c_1(t)\psi_1 + c_2\psi_2 \tag{15}$$

Equations 9, 10, and 11 yield the following equations as a result of the Schrödinger equation:

$$x(t) = c_1^*(t)c_2(t) + c_1(t)c_2^*(t) \tag{16}$$

$$y(t) = i[c_1^*(t)c_2(t) - c_1(t)c_2^*(t)] \tag{17}$$

$$z(t) = |c_2(t)|^2 - |c_1(t)|^2 \tag{18}$$

The final equation for $e(t)$ relates to an oscillating electric field interacting with the atom. For certain parameters, this system can exhibit chaotic behavior as will be demonstrated in the next section.

Implementation and Results

A general subroutine called `rk4` was written using pointers to arrays and dynamic memory to implement the 4^{th}-order 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. 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:

$$\frac{d^2y}{dt^2} + \omega^2 y = 0 \tag{19}$$

$$\frac{dy_0}{dt} = y_1 \tag{20}$$

$$\frac{dy_1}{dt} = -\omega^2 y_0 \tag{21}$$

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

$$y(t) = \cos(\omega t) \tag{22}$$

The `rk4` 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.

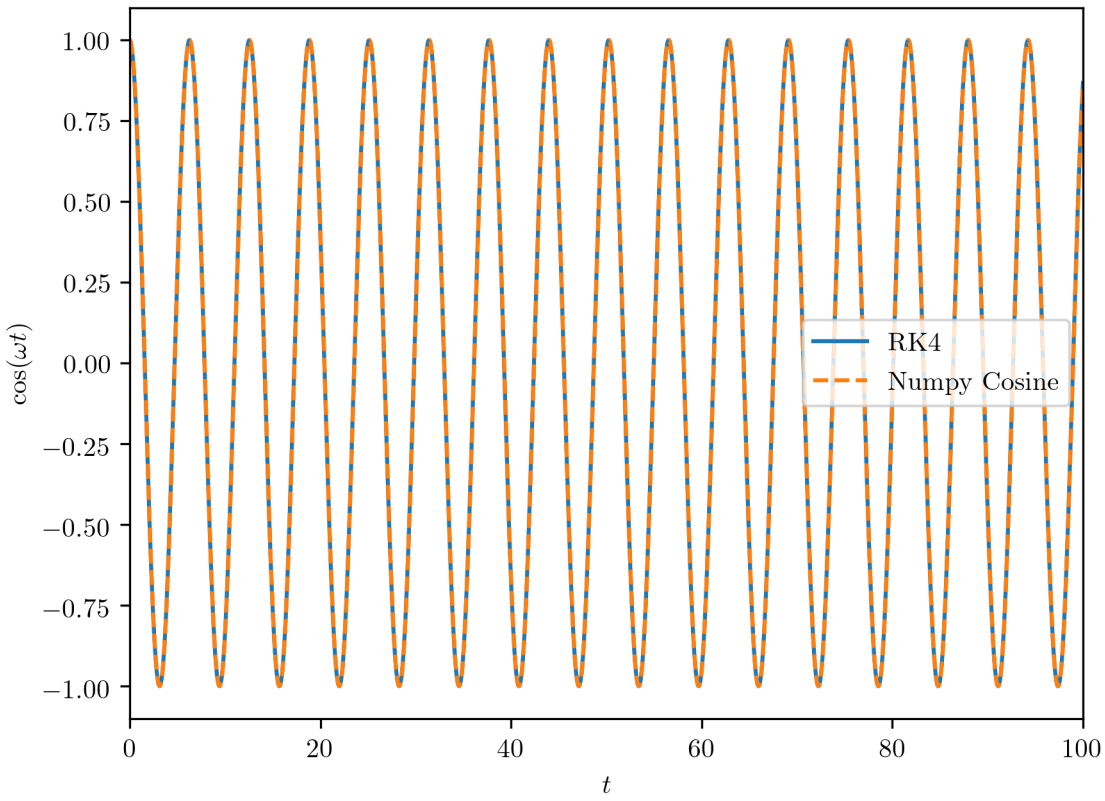

Figure 1: The cosine wave for the harmonic oscillator solution compared against Numpy's built-in cosine function, showing a high degree of accuracy.

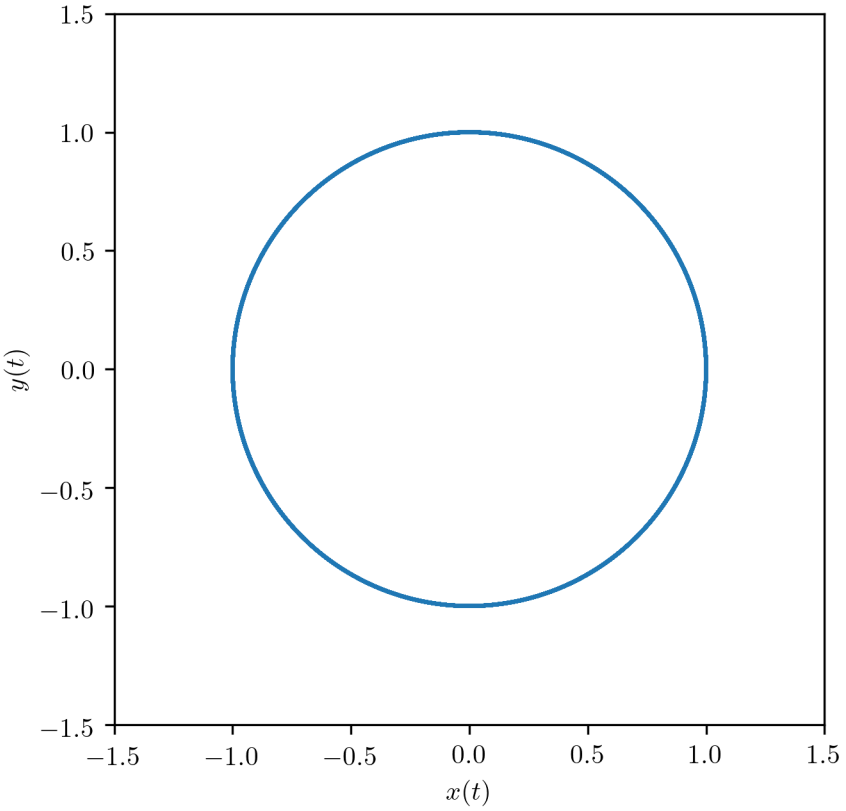

Figure 2: The phase space plot of the harmonic oscillator.

The `rk4` subroutine was used on the Jaynes-Cummings model after verification of correctness with Equations 9, 10, 11, 13, and 14 as inputs. The goal was to calculate $x(t)$, $y(t)$, $z(t)$, $e(t)$, and $e'(t)$ for $0 < t < 200$. To produce a stable solution, an adequate $h = \Delta t$ had to be chosen. Stability was analyzed by examining a plot of $x(t)$ vs. t for the domain of interest using different values of h. In total, $h = 1.0,\ 0.1,\ 0.01,$ and 0.001 were tested. The following initial conditions and parameters were used for testing:

$$\mu = \beta = 1 \tag{23}$$

$$x(0) = y(0) = e'(0) = 0 \tag{24}$$

$$z(0) = 1 \tag{25}$$

$$e(0) = 10^{-6} \tag{26}$$

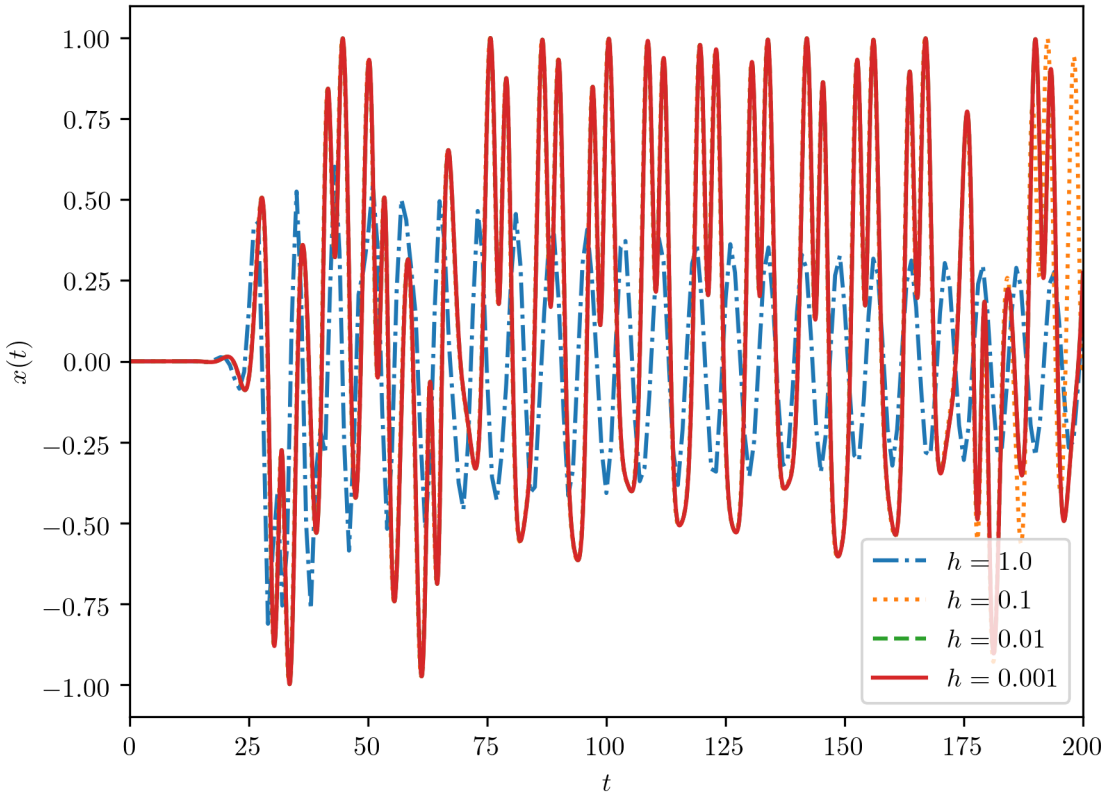

Figure 3: $x(t)$ vs. t for $h = 1.0,\ 0.1,\ 0.01,$ and 0.001.

From Figure 3, $x(t)$ stops changing shape in $0 < t < 200$ after $h = 0.01$, meaning $h = 0.01$ was a small enough step size to generate a stable solution. As such, $h = 0.01$ was used for all simulations. The following plots were generated from the data tabulated using the `rk4` subroutine.

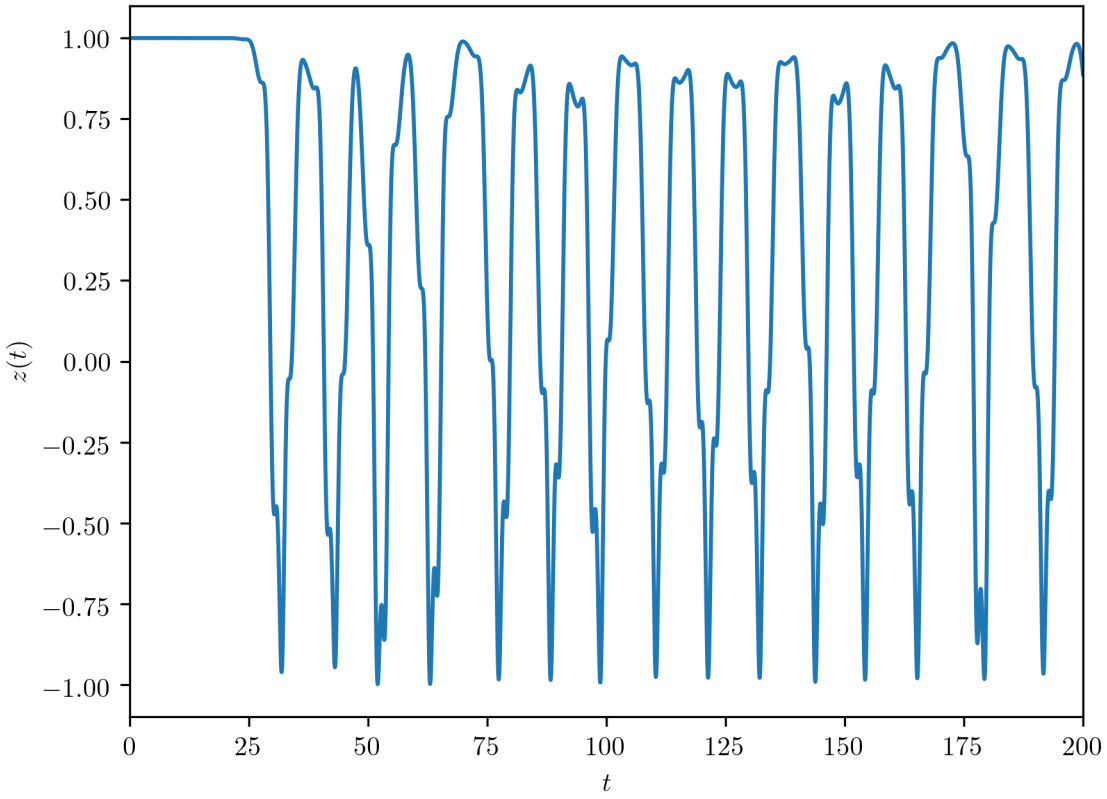

Figure 4: $z(t)$ vs. t plot for $h = 0.01$ using the stated initial conditions and parameter values.

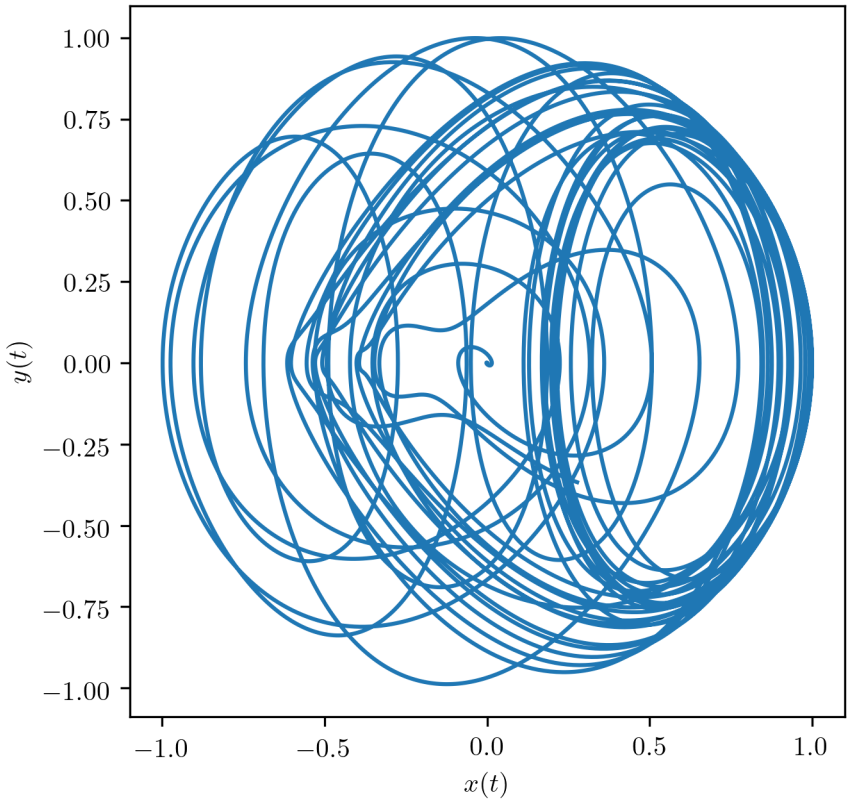

Figure 5: $y(t)$ vs. $x(t)$ plot for $h = 0.01$ using the stated initial conditions and parameter values.

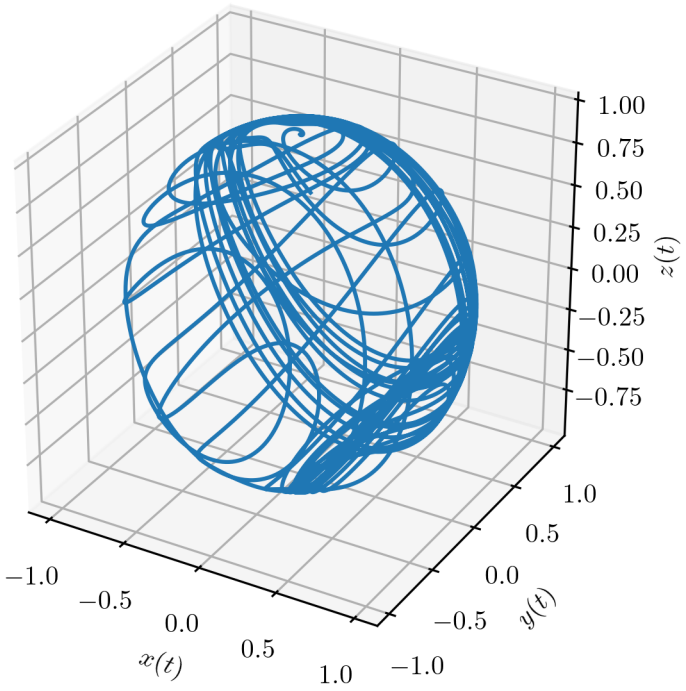

Figure 6: 3D plot of $x(t)$, $y(t)$, and $z(t)$ for $h = 0.01$ using the stated initial conditions and parameter values.

As expected, the system parameters and initial conditions yielded a solution that was chaotic in nature. From Figure 6, we can see that the system is bounded by a sphere showing no periodicity or recognizable pattern, which the other plots similarly suggest. When altering the value of β to 0, the following plots showed non-chaotic behavior.

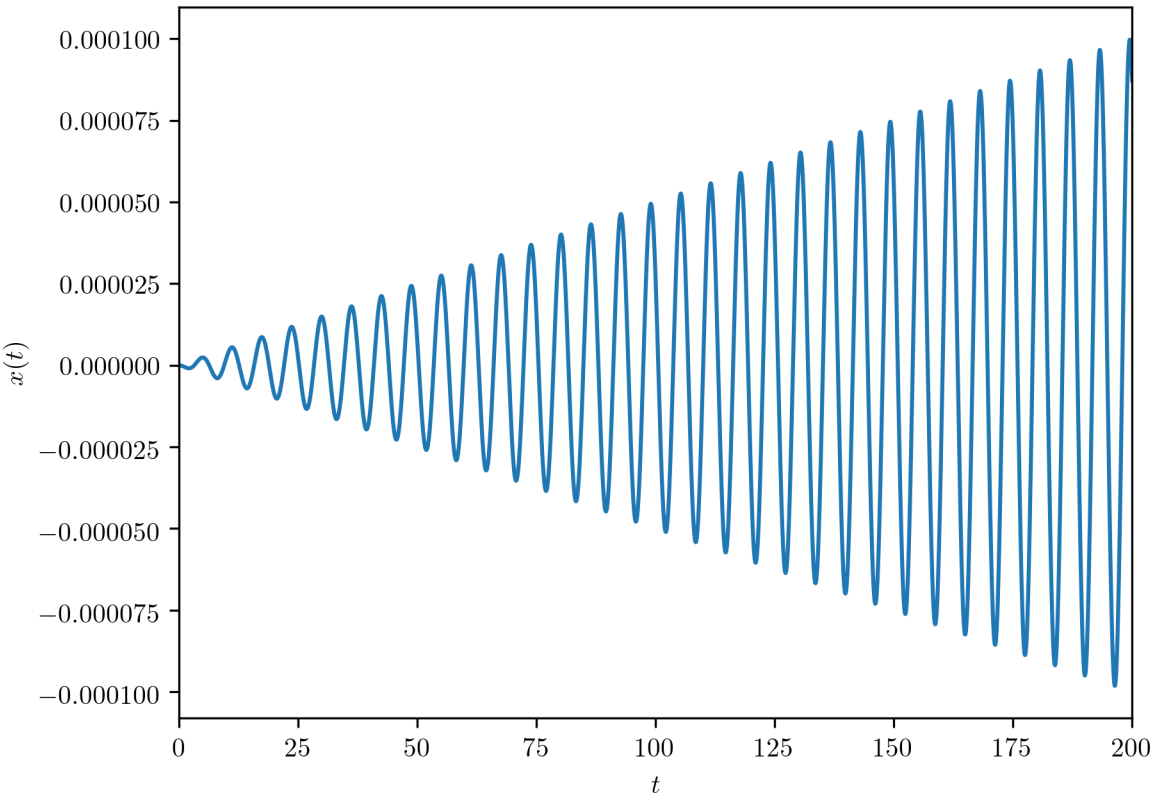

Figure 7: $x(t)$ vs. t for $h = 0.01$ using the stated initial conditions and $\beta = 0$.

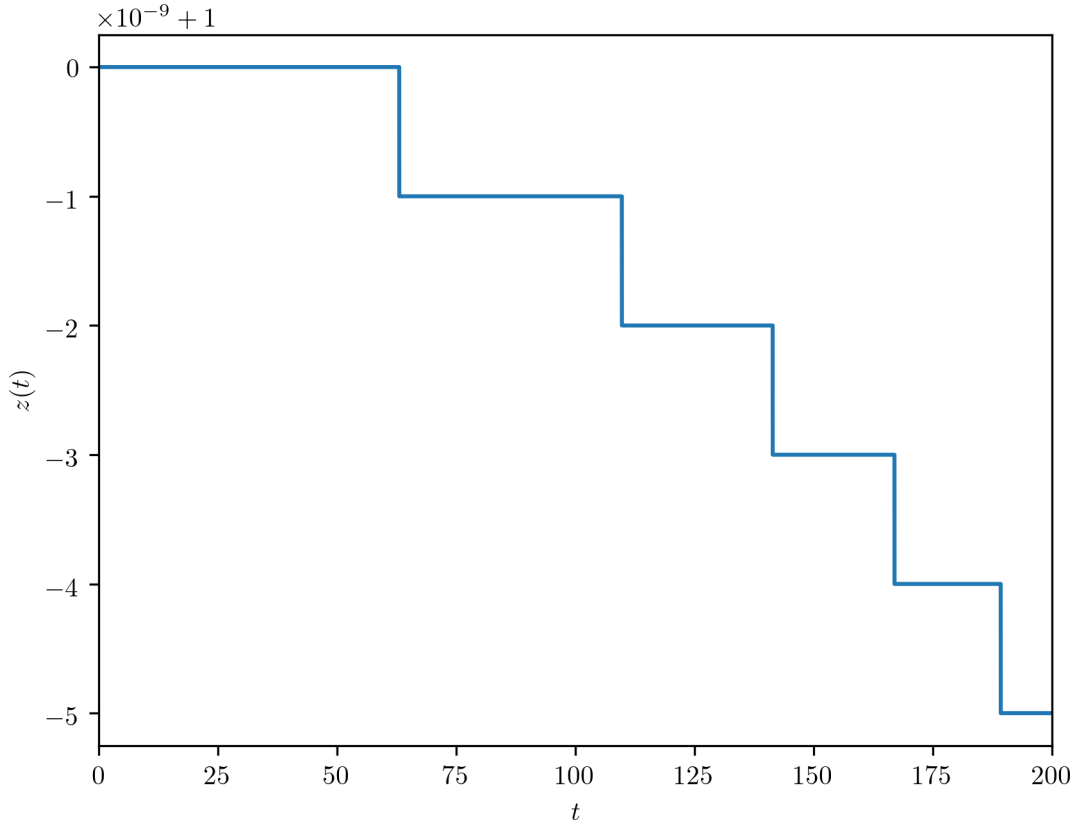

Figure 8: $z(t)$ vs. t plot for $h = 0.01$ using the stated initial conditions and $\beta = 0$.

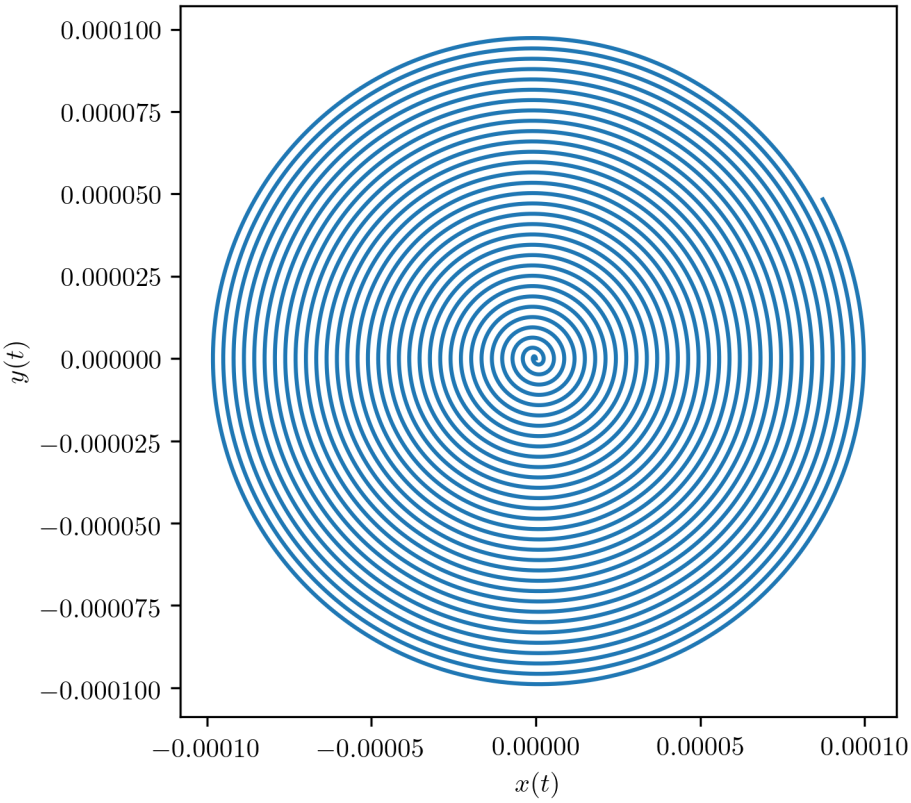

Figure 9: $y(t)$ vs. $x(t)$ plot for $h = 0.01$ using the stated initial conditions and $\beta = 0$.

A 3D plot for $\beta = 0$ was not generated since all values of $z(t)$ were within $\mathcal{O}(10^{-9})$ of each other, meaning $z(t)$ was essentially constant.

When altering the initial condition to $y(0) = 0.0001$, it was observed that the small change in the initial condition created drastically different behavior as expected of a system that exhibits chaotic behavior.

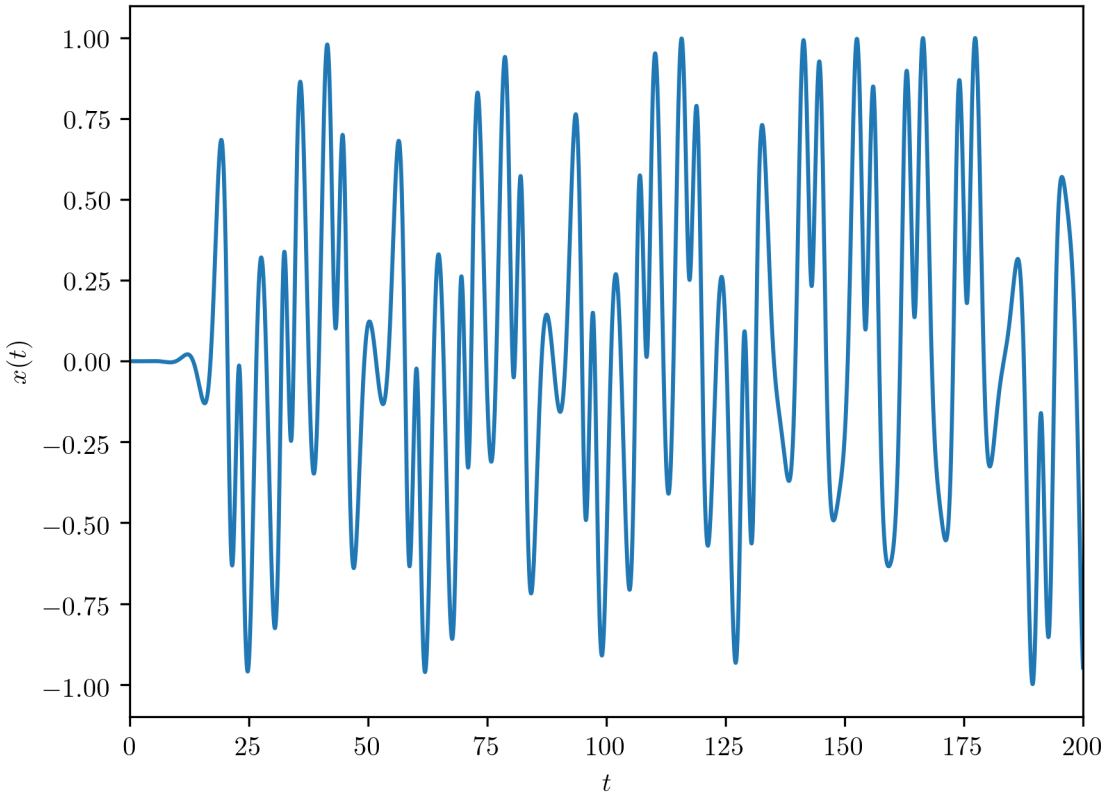

Figure 10: $x(t)$ vs. t using $y(0) = 0.0001$ and the stated parameters.

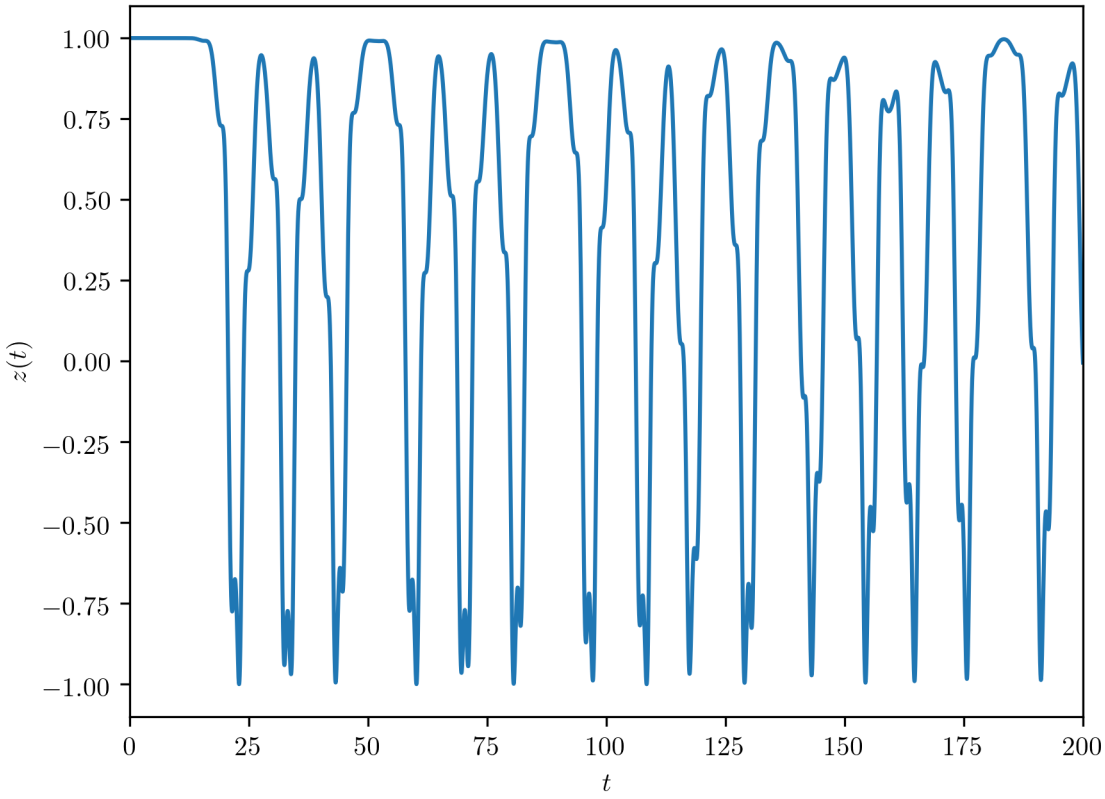

Figure 11: $z(t)$ vs. t plot for $h = 0.01$ using $y(0) = 0.0001$ and the stated parameters.

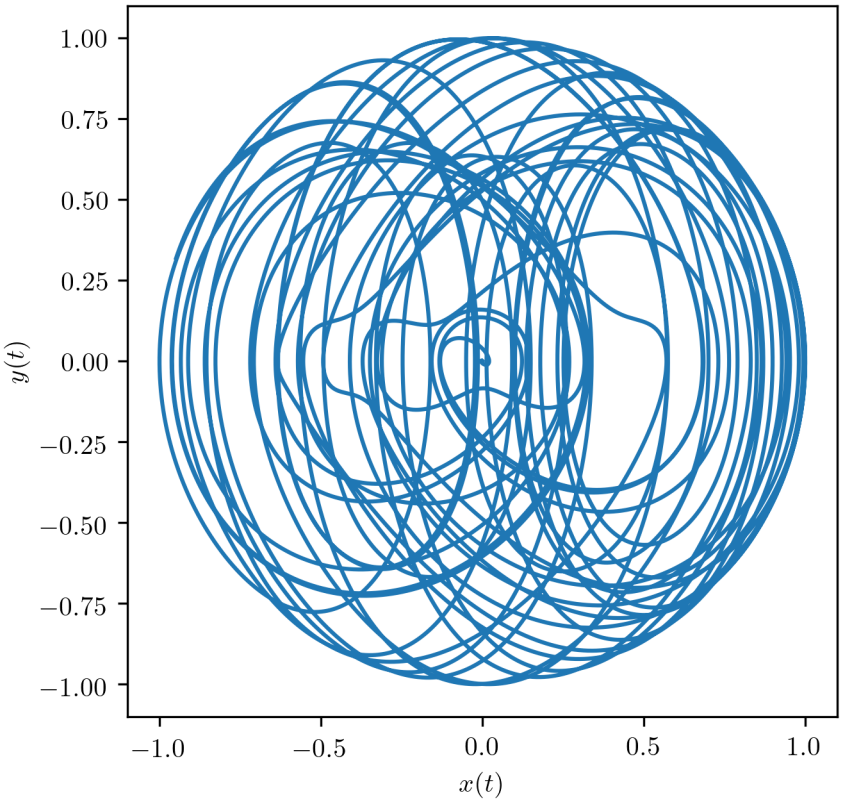

Figure 12: $y(t)$ vs. $x(t)$ plot for $h = 0.01$ using $y(0) = 0.0001$ and the stated parameters.

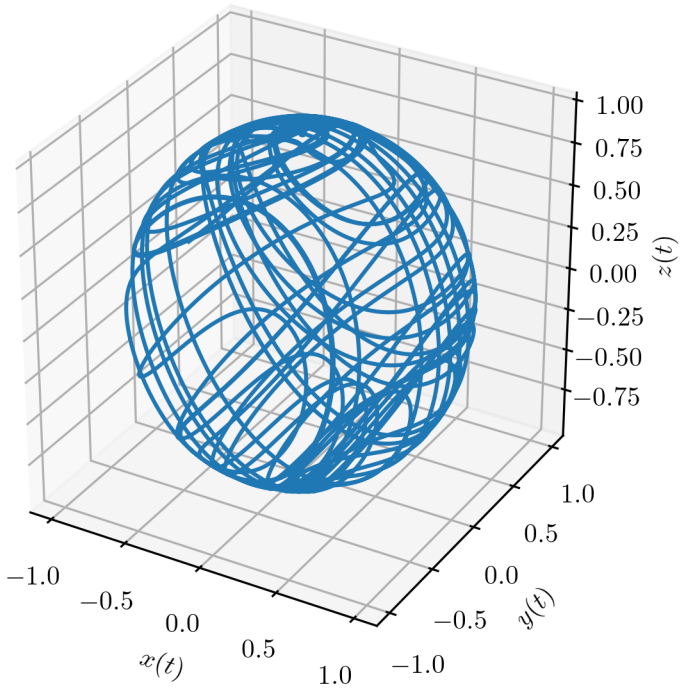

Figure 13: 3D plot of $x(t)$, $y(t)$, and $z(t)$ for $h = 0.01$ using $y(0) = 0.0001$ and the stated parameters.

Source Code

```
/* AEP 4380 Homework #3

Test ode solver
4th order Runge-Kutta is tested on a harmonic oscillator and chaotic system

Run on a core i7 using clang 1000.11.45.2 on macOS Mojave

Kevin Juan 3 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 rk4(double yold[], double ynew[], double h, double t, int N,
void frhs(double[], double, double[]))
{
    int i;
    double *k1, *k2, *k3, *k4, *temp;

    k1 = new double[5 * N];
    k2 = k1 + N;
    k3 = k2 + N;
    k4 = k3 + N;
    temp = k4 + N;

    frhs(yold, t, k1);
    for (i = 0; i < N; i++)
    {
        temp[i] = yold[i] + 0.5 * h * k1[i];
    }

    frhs(temp, t + 0.5 * h, k2);
    for (i = 0; i < N; i++)
    {
        temp[i] = yold[i] + 0.5 * h * k2[i];
    }

    frhs(temp, t + 0.5 * h, k3);
    for (i = 0; i < N; i++)
    {
        temp[i] = yold[i] + h * k3[i];
    }

    frhs(temp, t + h, k4);
    for (i = 0; i < N; i++)
    {
        ynew[i] = yold[i] + h * (k1[i] + 2.0 * (k2[i] + k3[i]) + k4[i])
        / 6.0;
    }

    delete[] k1;
    return;
```

```
}

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

void rhsJC(double y[], double t, double f[]) // Jaynes-Cummings rhs
{
    f[0] = -y[1];                          // dx/dt
    f[1] = y[0] + y[2] * y[3];             // dy/dt
    f[2] = -y[1] * y[3];                   // dz/dt
    f[3] = y[4];                           // de_o/dt
    f[4] = 1.0 * f[1] - 1.0 * 1.0 * y[3]; // de_1/dt
    return;
}

int main()
{
    int istep, nstep = 200000, N = 5, i;
    double yold[N], ynew[N], t, tinit = 0.0, h = 0.001;

    // // 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);
    // }

    // for (istep = 0; istep <= nstep; istep++)
    // {
    //     t = tinit + istep * h;
    //     rk4(yold, ynew, h, t, N, harmonicosc);
    //     fp1 << setw(20) << t << setw(20) << ynew[0] << setw(20) <<
```

```
    //     ynew[1] << endl;
    //     for (i = 0; i < N; i++)
    //     {
    //         yold[i] = ynew[i];
    //     }
    // }
    // fp1.close();

    // Runge-Kutta for Jaynes-Cummings
    yold[0] = 0.0;      // x
    yold[1] = 0.0;      // y
    yold[2] = 1.0;      // z
    yold[3] = 0.000001; // e
    yold[4] = 0.0;      // e'

    ofstream fp2;                     // output file using streams
    fp2.open("jaynes_cummings.dat"); // open new file for output
    fp2.precision(9);                 // select 9 digits

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

    for (istep = 0; istep <= nstep; istep++)
    {
        t = tinit + istep * h;
        rk4(yold, ynew, h, t, N, rhsJC);
        fp2 << setw(10) << t << setw(20) << ynew[0] << setw(20) << ynew[1]
        << setw(20) << ynew[2] << setw(20) << ynew[3] << setw(20) << ynew[4]
        << endl;
        for (i = 0; i < N; i++)
        {
            yold[i] = ynew[i];
        }
    }
    fp2.close();
    return (EXIT_SUCCESS);
}
```

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.

[2] Rubin H. Landau, Jose Paez, and Christian C. Bordeianu. *A Survey of Computational Physics: Introductory Computational Science*. Princeton University Press, Princeton, 2008.

[3] J.R. Ackerhalt, P.W. Milonni, and M.-L. Shih. Chaos in quantum optics. *Physics Reports*, 128(4-5):205–300, November 1985.