AEP4380 HW4
Ordinary Differential Equations and Chaotic Systems

Dan Girshovich

Feb 28, 2013

1 Overview

This document details the results of a C++ program which numerically solves a set of coupled ordinary differential equations (ODEs) that model a light wave in a system of two level atoms. These equations are solved using the fourth-order Runge-Kutta method as given in section 17.1 of Numerical Recipes[1]. The light wave model uses the Jaynes-Cummings equations, which are described in section 3 of Ackerhalt[2] et al. The parameters were chosen to produce a chaotic system, so a numerical solution is necessary.

2 Fourth-Order Runge-Kutta

The fourth-order Runge-Kutta method is defined in equation 17.1.2 of Numerical Recipes[1] as:

$$\begin{aligned}
k_1 &= hf(x_n, y_n) \\
k_2 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_1) \\
k_3 &= hf(x_n + \frac{1}{2}h, y_n + \frac{1}{2}k_2) \\
k_4 &= hf(x_n + h, y_n + k_3) \\
y_{n+1} &= y_n + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 + O(h^5)
\end{aligned}$$

Where h is the step size and $f(x_n, y_n) = \frac{dy}{dx}|_{x=x_n}$ is the first-order ODE being solved. The code for this is in the `rk4` function and has only been slightly modified from the Numerical Recipes version (the vector types were changed to all be the same, for simplicity). This method only requires the knowledge of y_n at one point and $f(x, y)$ in order to approximate the next value. So, given $f(x, y)$ and an initial condition on $y(x)$, `rk4` can be used to iteratively produce an approximate solution.

3 Simple Harmonic Oscillator Test

Before solving the light wave system, the code using `rk4` was first tested on the simple harmonic oscillator without damping. The ODE for this system is given by:

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

To use the Runge-Kutta method, this must be broken up into coupled first-order ODEs:

$$\frac{dy_0}{dt} = y_1$$
$$\frac{dy_1}{dt} = -\omega^2 y_0$$

The `gen_oscillator_data` function uses `rk4` to solve these in parallel via vectors. Given $\omega = 1.0$ and initial conditions $y(t = 0) = 1$, $y'(t = 0) = 0$, `gen_oscillator_data` produced the solution plotted below (using 1000 steps over three periods).

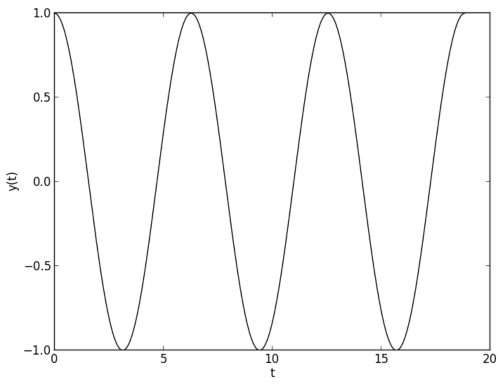


Figure 1: Simple harmonic oscillator solution over three periods (y(t) vs. t)

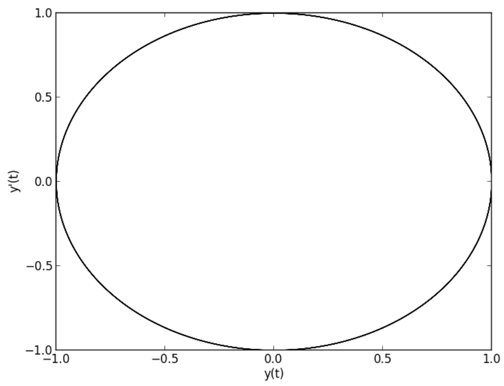


Figure 2: Simple harmonic oscillator phase plot (y'(t) vs. y(t))

As expected, the solution is a cosine function with a period of $\frac{2\pi}{\omega} = 2\pi$ and an amplitude of 1. The amplitude and derivative remained accurate up to at least seven significant figures over the three periods, as shown in the table below. The phase plot closes on itself, which implies that the phase also remained approximately constant. Note: $3\pi = 18.84955592$.

t	y(t)	y'(t)
0.00000000	1.00000000	0.00000000
0.01884956	0.99982235	-0.01884844
6.27690212	0.99998026	0.00628315
6.29575168	0.99992104	-0.01256603
12.55380424	0.99992104	0.01256605
12.57265380	0.99998026	-0.00628313
18.83070637	0.99982235	0.01884846
18.84955592	1.00000000	0.00000002

4 Light Wave Model

4.1 The Jaynes-Cummings Equations

The Jaynes-Cummings model for a light wave in a system of two level atoms is described in section 3 of Ackerhalt[2] et al. It has the following equations:

$$\begin{aligned}\frac{dx}{dt} &= -y\\ \frac{dy}{dt} &= x + ez\\ \frac{dz}{dt} &= -ey\\ \frac{d^2e}{dt^2} + \mu^2 e &= \beta\frac{dy}{dt}\end{aligned}$$

The last of these is not a first-order ODE, so it must be decomposed as follows:

$$\begin{aligned}\frac{de_0}{dt} &= e_1\\ \frac{de_1}{dt} &= \beta\frac{dy}{dt} - \mu^2 e_0\end{aligned}$$

These five first-order ODEs define the system, where $e = e_0$ and $e' = e_1$.

4.2 Initial Conditions

The light wave system to solve has the following initial conditions:

$$\begin{aligned}\mu &= \beta = 1\\ x(0) &= y(0) = e'(0) = 0\\ z(0) &= 1\\ e(0) &= 10^{-6}\end{aligned}$$

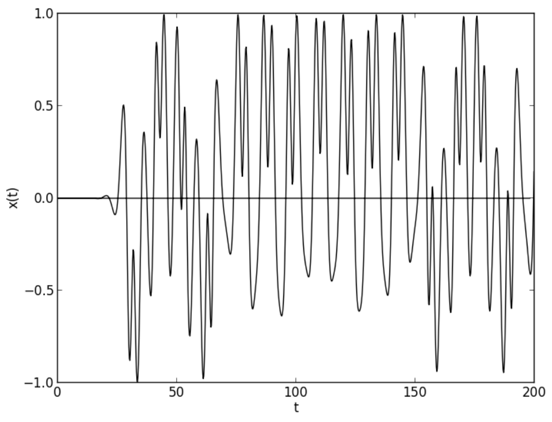
The initial conditions for e and e' need special treatment, because e has been decomposed into e_0 and e_1 in the coupled first-order ODEs. We see that:

$$\begin{aligned}e_0(0) &= e(0) = 10^{-6}\\ e_1(0) &= e_0'(0) = e'(0) = 0\end{aligned}$$

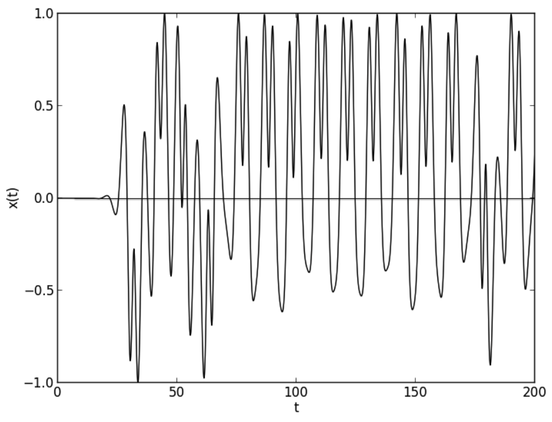
As required, each of the five coupled first-order ODEs describing the system has an initial condition.

4.3 Stable Solutions

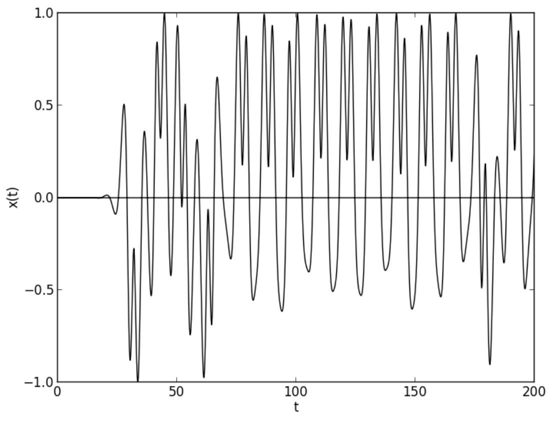
The function `gen_jc_data` uses `rt4` in a way similar to `gen_oscillator_data`, just with different initial conditions and coupled equations. Below are results for x(t) over the range $0 < t < 200$ with step sizing decreasing from left to right:



(a) h = 0.2



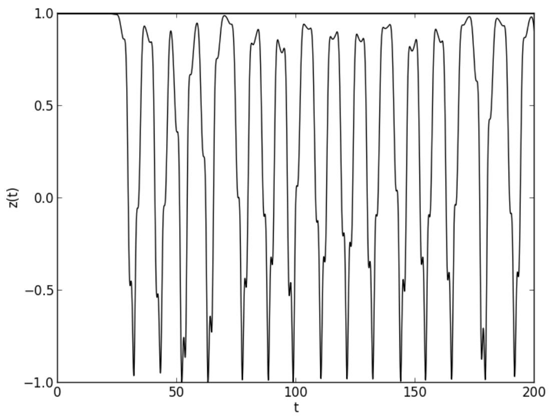
(b) h = 0.02



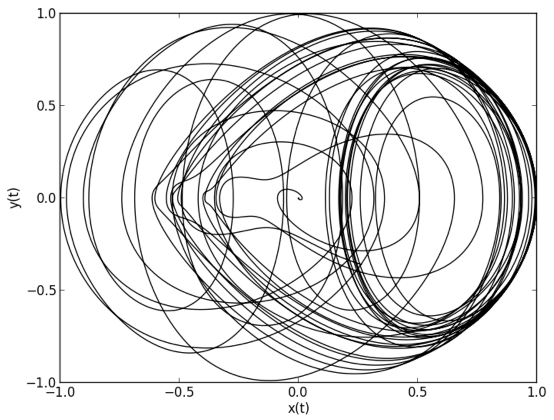
(c) h = 0.002

Figure 3: Three results for $x(t)$ at different step sizes

There are no discernible differences between the figures (b) and (c). Since each of these results were generated nearly instantly, no searching for a stable solution with a larger h was necessary - $h = 0.02$ should suffice. The results for $z(t)$ vs. t and $y(t)$ vs. $x(t)$ with the same step size are shown below.



(a) $z(t)$ vs. t



(b) $y(t)$ vs. $x(t)$

Figure 4: The results for $z(t)$ vs. t and $y(t)$ vs. $x(t)$ for $0 < t < 200$ with $h = 0.02$

Because the parameters given define a chaotic system, the small initial perturbation of $e(0) = 10^{-6}$ is enough to set off dramatic and unpredictable behavior for $t > 20$ (roughly). This means that over a large enough time scale, the compounded discretization error would stimulate a similar chaotic response and make the solution wildly inaccurate. The phase plot does not loop back over the same pattern, indicating there is no periodicity (as expected for a chaotic system). Also, the dark oval on the right side of the plot appears to indicate an attractor in that region.

5 Source Code

```
/*
AEP 4380 HW #4
Dan Girshovich
2/28/13
Generates plot data and root info related to Bessel functions.
Compile with: g++ -std=c++0x -O2 -o gen_data hw3.cpp
Tested on Mac OSX 10.8.2 with a Intel Core 2 Duo
*/

#include <cstdio>
#include <cstdlib>
#include <cmath>
#include "nr3.h"

// type / func defs
typedef void (*d_func)(const Doub, VecDoub &, VecDoub &);
void oscillator_derivs(const Doub, VecDoub &, VecDoub &);
void jc_derivs(const Doub, VecDoub &, VecDoub &);
void rk4(VecDoub &, VecDoub &, const Doub, const Doub, VecDoub &, d_func);
void gen_oscillator_data(const char *, double, double, int);
void gen_jc_data(const char *, double, double, int);
```

```
FILE *open_file_w(const char *);

const double w = 1.0; // oscillator frequency
const double mu = 1.0;
const double beta = 1.0;
const double pi = 4.0 * atan(1.0);

int main()
{
    gen_oscillator_data("oscillator_sol.dat", 0.0, 6 * pi, 1000);
    gen_jc_data("jc_sol_10000.dat", 0.0, 200.0, 10000);
}

// finds a solution to the simple harmonic oscillator equations between t1 and
// t2 using n steps of rk4.
void gen_oscillator_data(const char *fname, double t1, double t2, int n)
{
    FILE *fp = open_file_w(fname);
    const double h = (t2 - t1) / n;
    double t;
    VecDoub y(2);
    VecDoub dydt(2);
    // initial conditions
    y[0] = 1;
    y[1] = 0;
    oscillator_derivs(t1, y, dydt);
    // will hold values for next step
    VecDoub y_next(2);
    for (int i = 0; i < n; i++)
    {
        t = t1 + i * h;
        fprintf(fp, "%16.8f %16.8f %16.8f \n", t, y[0], dydt[0]);
        rk4(y, dydt, t, h, y_next, oscillator_derivs);
        //printf("%16.8f\n", y_next[0]);
        y = y_next; // uses overloaded assignment in NRvector
        oscillator_derivs(t, y, dydt);
    }
    fprintf(fp, "%16.8f %16.8f %16.8f \n", t2, y[0], dydt[0]);
    fclose(fp);
}

// returns the oscillator derivatives of each y at t in dydt
void oscillator_derivs(const Doub t, VecDoub &y, VecDoub &dydt)
{
    dydt[0] = y[1];
    dydt[1] = -1 * w * w * y[0];
}

// generates data for the plots of each parameter in the Janyes-Cummings model
void gen_jc_data(const char *fname, double t1, double t2, int n)
{
    FILE *fp = open_file_w(fname);
    const double h = (t2 - t1) / n;
    double t;
    VecDoub f(5); // x, y, z, e0, e1
    VecDoub dfdt(5); // x', y', z', e0', e1'
    // initial conditions
    f[0] = f[1] = 0; // x(0) = y(0) = 0
    f[2] = 1; // z(0) = 1
    f[3] = 1e-6; // e0(0) = 1e-6
    f[4] = 0; // e1(0) = e0'(0) = 0
    jc_derivs(t1, f, dfdt);
    // will hold values for next step
    VecDoub f_next(5);
    for (int i = 0; i < n; i++)
    {
        t = t1 + i * h;
        fprintf(fp, "%16.8f %16.8f %16.8f %16.8f %16.8f %16.8f \n",
            t, f[0], f[1], f[2], f[3], dfdt[3]); // t, x, y, z, e0, e0'
        rk4(f, dfdt, t, h, f_next, jc_derivs);
        f = f_next; // uses overloaded assignment in NRvector
        jc_derivs(t1, f, dfdt);
    }
    // safer to print manually instead of looping past what the input specifies
    fprintf(fp, "%16.8f %16.8f %16.8f %16.8f %16.8f %16.8f \n",
        t2, f[0], f[1], f[2], f[3], dfdt[3]); // t, x, y, z, e0, e0'
    fclose(fp);
}

// returns the Jaynes-Cummings derivatives of each y at t in dydt
void jc_derivs(const Doub t, VecDoub &f, VecDoub &dfdt)
{
    dfdt[0] = -1 * f[1]; // x' = - y
    dfdt[1] = f[0] + f[3] * f[2]; // y' = x + ez
    dfdt[2] = -1 * f[3] * f[1]; // z' = -ey
    dfdt[3] = f[4]; // e0' = e1
    dfdt[4] = beta * dfdt[1] - mu * mu * f[3]; // e1' = By' - u^2 e0
}

// Numerical Recipes p909 - slightly modified
void rk4(VecDoub &y, VecDoub &dydt, const Doub t, const Doub h,
         VecDoub &yout, d_func derivs)
{
    Int n = y.size();
    VecDoub dym(n), dyt(n), yt(n);
    Doub hh = h * 0.5;
    Doub h6 = h / 6.0;
    Doub th = t + hh;
    for (Int i = 0; i < n; i++) yt[i] = y[i] + hh * dydt[i];
    derivs(th, yt, dyt);
    for (Int i = 0; i < n; i++) yt[i] = y[i] + hh * dyt[i];
    derivs(th, yt, dym);
    for (Int i = 0; i < n; i++)
    {
        yt[i] = y[i] + h * dym[i];
        dym[i] += dyt[i];
    }
    derivs(t + h, yt, dyt);
    for (Int i = 0; i < n; i++)
        yout[i] = y[i] + h6 * (dydt[i] + dyt[i] + 2.0 * dym[i]);
}


// opens a new file for output
FILE *open_file_w(const char *fname)
{
    FILE *fp;
    fp = fopen(fname, "w+");
    if (NULL == fp)
    {
        printf("cannot open file\n");
        return (EXIT_SUCCESS);
    }
    return fp;
}
```

References

[1] W.H. Press, S.A Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipies, The Art of Scientific Computing.* Camb. Univ. Press, 3rd Edition, 2007.

[2] J. R. Ackerhalt, P. W. Milonni and M.-L. Shih *Chaos in Quantum Optics.* Physics Reports Vol. 128, 1985, pages 205-300.