PHY 410
Homework Assignment 7

Han Wen

Person No. 50096432

November 3, 2014

Abstract

The goal of this assignment was to get familiar with ODE computation and realize the application on non-linear physics.

Contents

1 Problem 1

1.1 Description

Using inspiration from standard PHY 301-302 Intermediate Mechanics and PHY 509 Classical Dynamics textbooks, generate several trajectories of the nonlinear driven pendulum for the parameters suggested in your mechanics textbook. Generate a bifurcation diagram (as described in class, and here `http://en.wikipedia.org/wiki/Bifurcation_diagram`) for a few values of the input strength between 1 and 1.1. Your choice should capture some of the relevant dynamics (i.e. you should have examples of one-cycles, two-cycles, four-cycles, chaotic cycles).

You can also use this reference for inspiration :

`http://www.thphys.uni-heidelberg.de/~gasenzer/index.php?n1=teaching&n2=chaos`

Hints : (a) You may have to protect against numerical division by zero here (*) for some values of parameters :

AvePeriod /= nPeriod
for i in xrange(nPeriod) :
ErrorBar += (period[i] - AvePeriod)*(period[i] - AvePeriod)
if $nPeriod > 1$:
ErrorBar = sqrt(ErrorBar/(nPeriod*(nPeriod-1)))
else :
ErrorBar = 0.0
print "Average period = " + str(AvePeriod) + " +/- " + str(ErrorBar)

(b) You should throw away 100 cycles in the Poincare map to get appropriate dynamics.

1.2 Numerical Analysis

The result is shown in the diagram: Fig. 1. From the plot we can observe the required four phases. With D_F from 1 to about 1.01, it is 1-cycles case; from about 1.01 to about 1.06, it is 2-cycles case; from 1.06 to about 1.07 it is 4-cycles case; from that point on it is chaotic cycles until about 1.08.

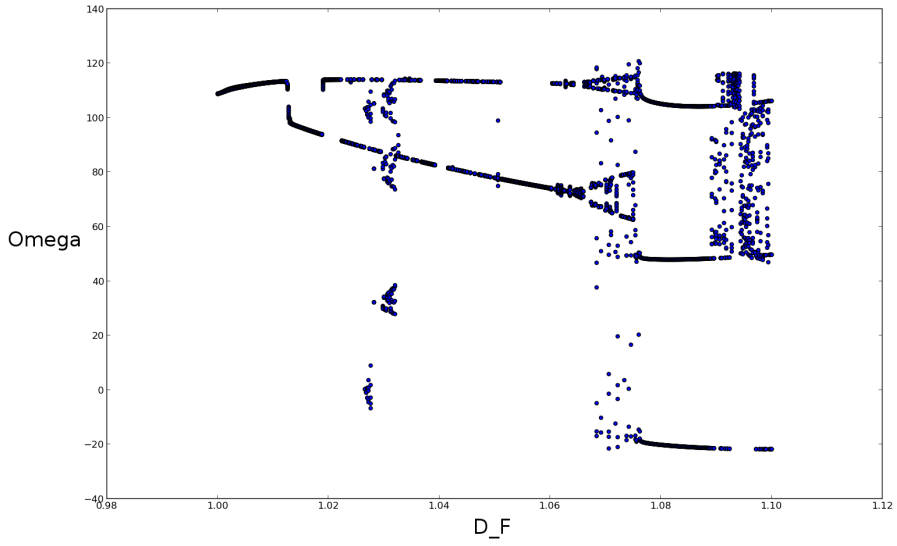

Figure 1: bifurcation

2 Problem 2

2.1 Description

In class we considered the Lorenz model :

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = x(\rho - z) - y$$

$$\frac{dz}{dt} = xy - \beta z$$

with sigma, rho, and beta constants (chosen as sigma=10, beta=8/3 ,and rho = 28).

Write software to solve this problem with both the Runge-Kutta 4-th order procedure, and the RK4 Adaptive stepsize procedure. Compare and contrast the convergence and computational time to solve the problem. Is the adaptive stepsize procedure worthwhile in this case? Also demonstrate the butterfly effect, whereby two very close initial points can diverge after N iterations. Demonstrate this graphically (for instance, by showing the two cycles when they diverge, or by plotting the difference of the trajectories versus the time step).

2.2 Result

Inspired by enormous glorious achievement made by our predecessors with the one of the oldest yet most long-lasting language, Fortran! as well as the ambition to be a qualified computational-physicist, I decide to use Fortran and write the origin code of RK method. Here are the result:
For initial coordinates(10,10,10) the diagram: Fig 2
For initial coordinates(10,10,10.1) the diagram: Fig 3

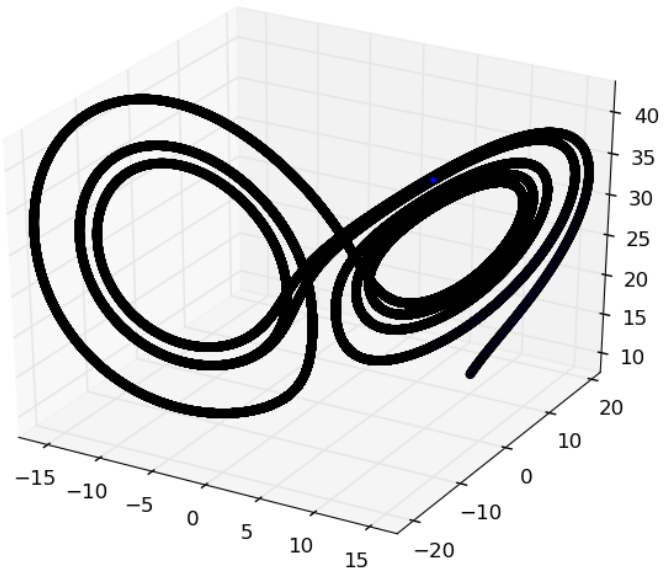

Figure 2: butterfly 10 10 10

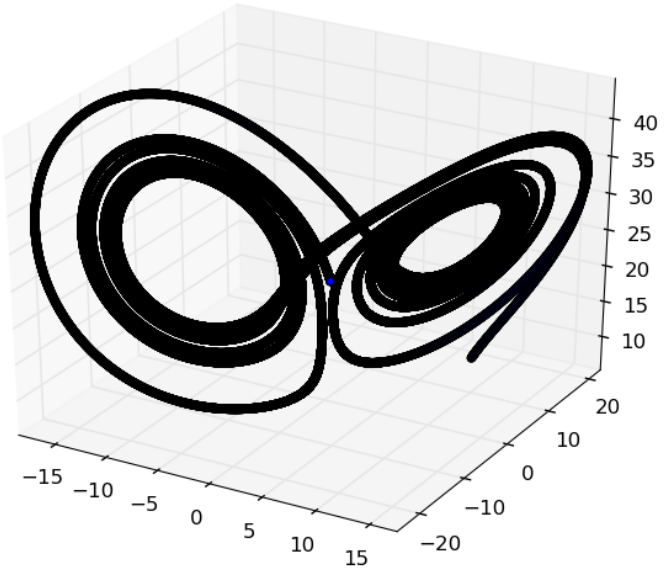

Figure 3: butterfly 10 10 10.1

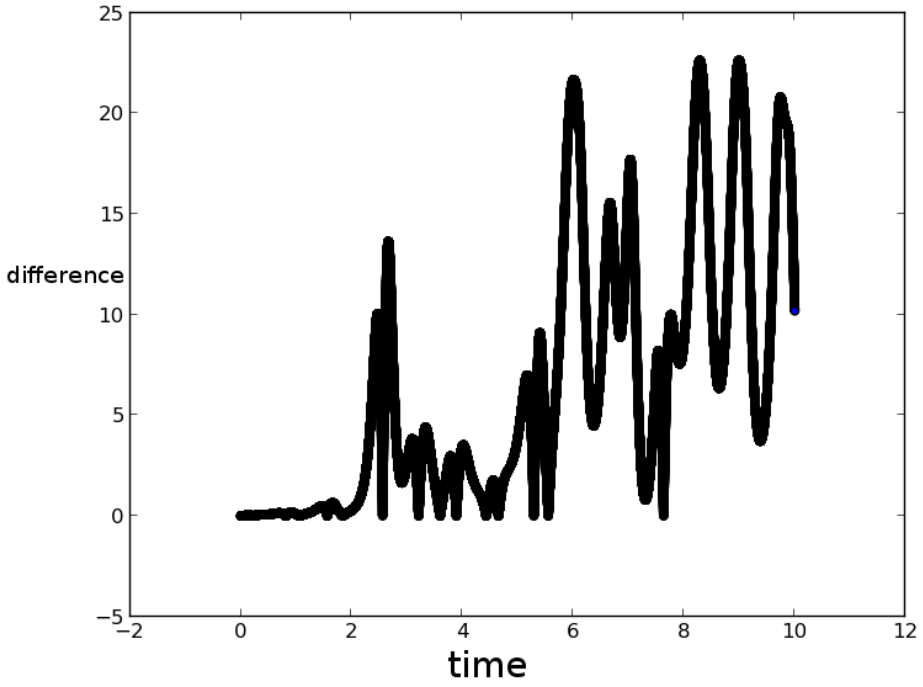

Figure 4: difference of x component

We can see although they two are quite similar, but still fundamentally different. Here is the difference of component vs. time : Fig 4

Since I am using Fortran, using adaptive method may not be a good choice and the comparison is somehow pointless. Since for Fortran, as far as I know, floating-point calculation will be much faster than other language while other syntax such as if will not be that fast. To simulate for same amount of time, regular RK-4 method took about 0.33 s, while adaptive method took 3.69 s, despite I gave a rather big accuracy, and consequently it generate much more data. Therefore I should say the adaptive method is not worthwhile and Fortran is a really good language for physicist if they want to do some simple structured yet highly FLoating-point Operational computing.
By the way, I think usually the adaptive method is referring to different order of RK method.

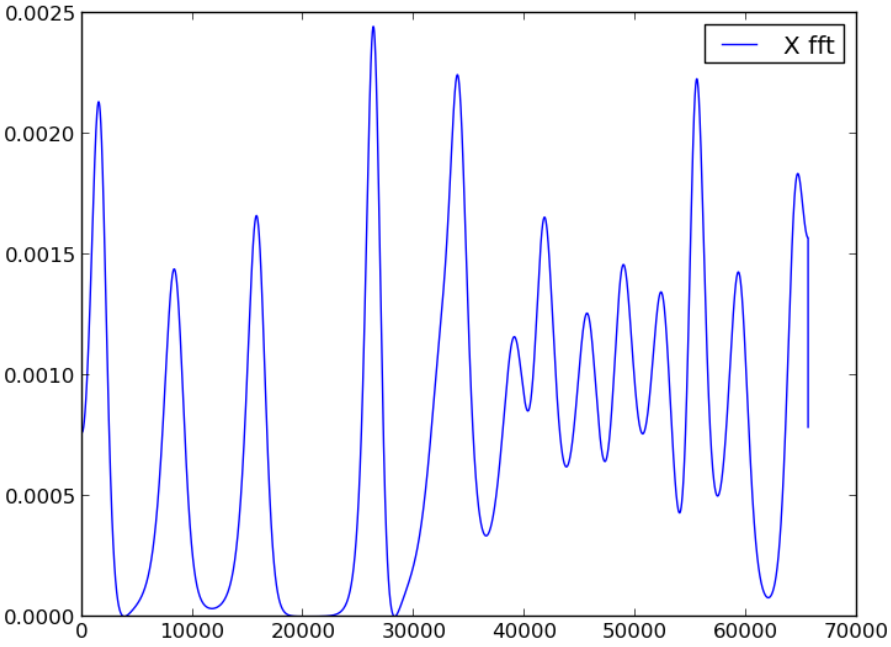

Figure 5: FFT power spectrum of x component

3 Problem 3

3.1 Description

Generate the FFT power spectrum for the x, y, and z time series in Problem 2. Plot them. Compare the spectra for similarities and differences. What does this tell you about the behavior of the system?

3.2 Result

Here are the diagram of the fft of x,y,z component vs. time:Fig 5 6 7

We can see, first of all, these spectra are very similar, going up and down as frequency increases. Yet, they are more or less different in details, especially that of z compared to x and y. This tells us the system distribute the energy with frequency also randomly and differently in different direction, indicating the heterogeneous chaotic behaviour of the system in space.

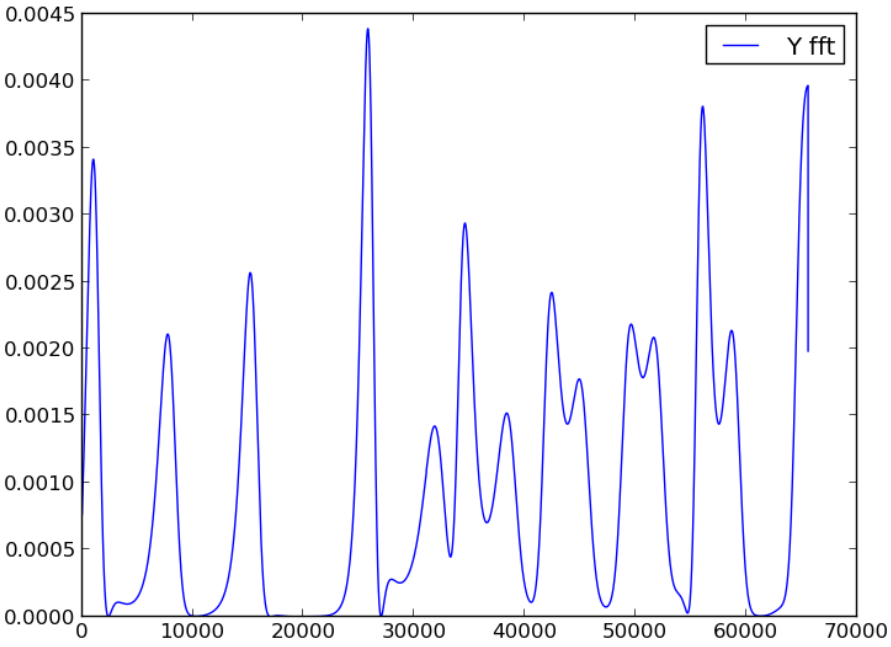

Figure 6: FFT power spectrum of y component

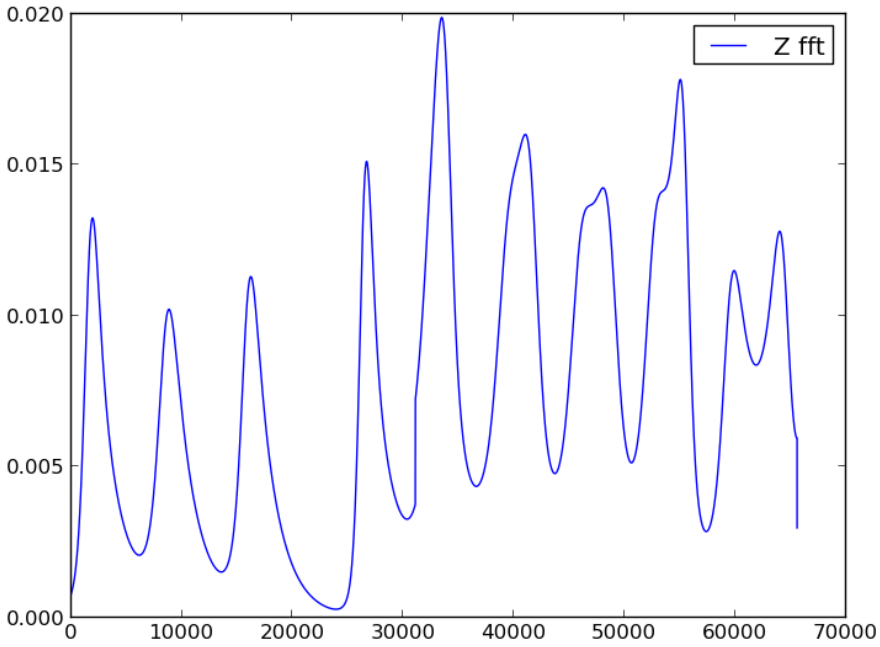

Figure 7: FFT power spectrum of z component

Acknowledgements

I discussed this assignment with my classmates and used material from the cited references, but this write-up is my own.

References

[1] PHY 410-505 Webpage, `http://www.physics.buffalo.edu/phy410-505`.

[2] RK method `http://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods#Adaptive_Runge.E2.80.93Kutta_methods`

[3] `http://www.thphys.uni-heidelberg.de/~gasenzer/index.php?n1=teaching&n2=chaos`

A Appendix

A.1 c++ fortran python code

The following python code was used to obtain the results in this report:

```
// pendul - Program to compute the motion of a simple pendulum
// using the Euler or Verlet method
#include <iostream>
#include <fstream>
#include <math.h>
#include <vector>
#include "nonlin.hpp"
#include "diffeq.hpp"

using namespace std;
using namespace cpt;

class PendulumFunction {
public :
  PendulumFunction( double iL=g, double igamma=0, double iF_D = 0, double iomega_D
    L(iL), omega_0( sqrt(g/L) ),gamma(igamma),F_D(iF_D),omega_D(iomega_D)
  {
  }

  static const double g=9.0;          // acceleration of gravity

protected :
  double L;                           // length in meters

  double omega_0;                     // natural frequency
  double gamma;                       // damping constant
  double F_D;                         // driving force amplitude
  double omega_D;                     // driving force frequency

};

class PendulumAcceleration : public PendulumFunction {
public :

  PendulumAcceleration( double iL=g, double igamma=0, double iF_D = 0, double iomeg
    PendulumFunction( iL, igamma, iF_D, iomega_D ) {}


  // Acceleration of the pendulum
  double operator() (Matrix<double,1> const & p ) const {
    double t=p[0];
    double theta=p[1];
    double dtheta_dt=p[2];
```

```
    double a = - omega_0 * omega_0 * sin(theta);      // due to gravity
    if ( gamma > 0.0 )
      a += - gamma * dtheta_dt;                        // damping
    if ( F_D > 0.0 )
      a += F_D * cos(omega_D * t);                     // driving force
    return a;
  }
};

class PendulumDiffEq : public PendulumFunction {
public :

  PendulumDiffEq( double iL=g, double igamma=0, double iF_D = 0, double iomega_D =
    PendulumFunction( iL, igamma, iF_D, iomega_D ) {}

  Matrix<double,1> operator()( Matrix<double,1> const & p) const {
    double t = p[0], x = p[1], y = p[2];

    Matrix<double,1> out(3);
    out[0] = 1.0;
    out[1] = y;
    out[2] = -omega_0*omega_0*sin(x) - gamma*y + F_D*cos(omega_D*t);

    return out;
  }

};



int main() {
  //* Select the numerical method to use: Euler or Verlet
  //cout << "Choose a numerical method : Euler (0), RK4 (1), or Adaptive RK4 (2): "
  //int method; cin >> method;
  int method = 1;
  //* Set initial position and velocity of pendulum
  //cout << "Enter initial angle (in degrees): ";
  //double theta0; cin >> theta0;
  double theta0=3.0;
  const double pi = 3.141592654;
  double theta = theta0*pi/180;   // Convert angle to radians
  //cout << "Enter initial angular frequency (degrees/s): ";
  //double omega;  cin >> omega;           // Set the initial velocity
  double omega = 0.0 ;
  omega = omega*pi/180.0;
  //cout << "Enter damping coefficient : ";
  //double gamma; cin >> gamma;
  double gamma = 0.5;
```

```
  //cout << "Enter driving amplitude : ";
  //double F_D; cin >> F_D;
  double F_D;                                   //VARYING THIS VALUE TO GET BIFURCATE
  //cout << "Enter driving frequency (degrees/s): ";
  //double omega_D; cin >> omega_D;
  double omega_D = 0.6666667;
  //omega_D = omega_D*pi/180.0;

  //* Set the physical constants and other variables
  double L = PendulumDiffEq::g;  // The constant g/L
  double time = 0.0;              // Initial time
  double time_old;                // Time of previous reversal
  int irev = 0;                   // Used to count number of reversals
  //cout << "Enter time step: ";
  double tau; // cin >> tau;
  tau = 0.003;

  ofstream biout("bifurcation.data");
  for( F_D = 1.0 ; F_D<1.1 ; F_D+=0.0002) {
//to calculate bifurcation by changing F_D


  PendulumAcceleration pendulumAcc( L, gamma, F_D, omega_D);

  PendulumDiffEq pendulumDiffEq( L, gamma, F_D, omega_D);

  double T_D = 2*pi/omega_D;       // Period of driving force

  double T_p = 0.0;                // Period for Poincare section

  if ( omega_D > 0.0 )
    T_p = T_D;
  else
    T_p = 2*pi/sqrt(PendulumDiffEq::g / L);


  double accuracy = 1e-6;

  //* Take one backward step to start Verlet

  Matrix<double,1> xv(3);
  xv[0] = time;
  xv[1] = theta;
  xv[2] = omega;
  double accel = pendulumAcc( xv );
  double theta_old = theta - omega*tau + 0.5*tau*tau*accel;
```

```
xv[0] = time;
xv[1] = theta_old;
xv[2] = omega;

int iperiod = 0;
int iperiod_old = -1;

//* Loop over desired number of steps with given time step
//    and numerical method
//cout << "Enter number of time steps: ";
int nStep;//  cin >> nStep;
nStep = 100000;                //giving a total of 500s enough to generate data

bool plotTrajectory = false; // plot Poincare and trajectory if true.
                             // plot only Poincare if false.

// if ( nStep > 10000 ) {
//   cout << "::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
//   cout << "::::: Warning : Only plotting Poincare section, not trajectory :::
//   cout << "::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
// }
// plotTrajectory = ( nStep < 10000 );

std::vector<double> t_plot;
std::vector<double> th_plot;
std::vector<double> om_plot;
std::vector<double> period;
std::vector< std::pair<double,double> > poincare_map;
double dt_min = tau;
double dt_max = tau;
int iStep;
for( iStep=0; iStep<nStep; iStep++ ) {


  theta_old = xv[1];

  //* Record angle and time for plotting
  if ( plotTrajectory ) {
    t_plot.push_back( xv[0] );

    double thplot = xv[1];
    double omplot = xv[2];

    while (thplot > pi ) thplot -= 2*pi;
    while (thplot <= -pi) thplot += 2*pi;

    th_plot.push_back( thplot*180/pi );   // Convert angle to degrees
    om_plot.push_back( omplot*180/pi );
```

```
    }

    iperiod = static_cast<int>( xv[0] / T_p );
    // Calculate a Poincare map of the dynamics
    if ( iperiod != iperiod_old ) {
      iperiod_old = iperiod;

      double thplot = xv[1];
      double omplot = xv[2];

      while (thplot > pi ) thplot -= 2*pi;
      while (thplot <= -pi) thplot += 2*pi;


      poincare_map.push_back( std::pair<double,double>(thplot*180/pi, omplot*180/pi
      std::cout << "Period reached. theta = " <<  thplot*180/pi << ", omega = " <<
omplot*180/pi << endl;
    }


    if ( method == 0 ){
      cpt::Euler_step( xv, tau, pendulumAcc );
    }
    else if( method == 1 ) {
      cpt::RK4_step( xv, tau, pendulumDiffEq );
    } else if ( method == 2) {

      tau = cpt::RK4_adaptive_step(xv, tau, pendulumDiffEq, accuracy);
      dt_min = min(tau, dt_min);
      dt_max = max(tau, dt_max);
    }

   //* Test if the pendulum has passed through theta = 0;
    //    if yes, use time to estimate period
    if( xv[1]*theta_old < 0 ) { // Test position for sign change
      //cout << "Turning point at time t = " << time << endl;
      if( irev == 0 )           // If this is the first change,
        time_old = xv[0];        // just record the time
      else {
        period.push_back( 2*(xv[0] - time_old) );
        time_old = xv[0];
      }
      irev++;        // Increment the number of reversals
    }
  }


  int nPeriod = irev-1;     // Number of times period is measured
```

```
  //* Estimate period of oscillation, including error bar
  double AvePeriod = 0.0, ErrorBar = 0.0;
  int i;
  for( i=0; i<nPeriod; i++ ) {
    AvePeriod += period[i];
  }
  AvePeriod /= nPeriod;
  for( i=0; i<nPeriod; i++ ) {
    ErrorBar += (period[i] - AvePeriod)*(period[i] - AvePeriod);
  }
  ErrorBar = sqrt(ErrorBar/(nPeriod*(nPeriod-1)));
  cout << "Average period = " << AvePeriod << " +/- " << ErrorBar << endl;

  //* Print out the plotting variables: t_plot, th_plot
  if ( plotTrajectory ) {
    ofstream plotOut("pend_plot.txt");
    for( i=0; i< t_plot.size(); i++ ) {
      plotOut << t_plot[i] << "  " << th_plot[i] << " " << om_plot[i] << endl;
    }
    plotOut.close();
  }

  int wh;
  //ofstream poincareOut("poincare_plot.txt");
  for( wh=10; wh< 30; wh++ ) {
    //poincareOut << poincare_map[i].first << " " << poincare_map[i].second << en
    biout << F_D << " " << poincare_map[wh].second << endl;

  }
};
  biout.close();
  return 0;
}
```

```
program WH_BUTTERFLY
real,external :: f1,f2,f3
real :: k1,k2,k3,k4,l1,l2,l3,l4,m1,m2,m3,m4,x,y,z,t
real :: x0,y0,z0
real,parameter :: h=0.0001
real :: t1,t2
integer :: i,n
open (unit=8,file='wh_butterfly.txt')
t=0
n=1000000
write(*,*)"give x0,y0,z0"
read(*,*)x0,y0,z0
call cpu_time(t1)
```

```
do i=1,n,1
write(8,*) x0,y0,z0
k1=f1(x0,y0)
l1=f2(x0,y0,z0)
m1=f3(x0,y0,z0)
k2=f1(x0+h/2*k1,y0+h/2*l1)
l2=f2(x0+h/2*k1,y0+h/2*l1,z0+h/2*m1)
m2=f3(x0+h/2*k1,y0+h/2*l1,z0+h/2*m1)
k3=f1(x0+h/2*k2,y0+h/2*l2)
l3=f2(x0+h/2*k2,y0+h/2*l2,z0+h/2*m2)
m3=f3(x0+h/2*k2,y0+h/2*l2,z0+h/2*m2)
k4=f1(x0+h*k3,y0+h*l3)
l4=f2(x0+h*k3,y0+h*l3,z0+h*m3)
m4=f3(x0+h*k3,y0+h*l3,z0+h*m3)
x=x0+1/6.0*(k1+2*k2+2*k3+k4)*h
y=y0+1/6.0*(l1+2*l2+2*l3+l4)*h
z=z0+1/6.0*(m1+2*m2+2*m3+m4)*h
t=t+h
x0=x
y0=y
z0=z
end do
call cpu_time(t2)
write(*,*)"THE RESULT HAS BEEN WRITEN IN THE TXT FILE"
write(*,*)t2-t1
stop
end

real function f1(x,y)
real x,y
real :: a=10.0
f1=a*(y-x)
return
end

real function f2(x,y,z)
real x,y,z
real :: c=28
f2=c*x-y-x*z
return
end

real function f3(x,y,z)
real x,y,z
real :: b=8.0/3.0
f3=-b*z+x*y
return
end
```

```
program WH_BUTTERFLYi
!adaptive method
real,external :: f1,f2,f3
real :: k1,k2,k3,k4,l1,l2,l3,l4,m1,m2,m3,m4,x,y,z,t
real :: k21,k22,k23,k24,l21,l22,l23,l24,m21,m22,m23,m24
real :: x0,y0,z0
real :: acc
real :: h
real :: t1,t2
integer :: i,n
open (unit=8,file='adwh_butterfly.txt')
t=0
!n=100000
acc=0.000001
write(*,*)"give x0,y0,z0"
read(*,*)x0,y0,z0
!do i=1,n,1
call cpu_time(t1)
do while(t<10.0)
write(8,*) x0,y0,z0
h=0.001

100 k1=f1(x0,y0)
l1=f2(x0,y0,z0)
m1=f3(x0,y0,z0)
k2=f1(x0+h/2*k1,y0+h/2*l1)
l2=f2(x0+h/2*k1,y0+h/2*l1,z0+h/2*m1)
m2=f3(x0+h/2*k1,y0+h/2*l1,z0+h/2*m1)
k3=f1(x0+h/2*k2,y0+h/2*l2)
l3=f2(x0+h/2*k2,y0+h/2*l2,z0+h/2*m2)
m3=f3(x0+h/2*k2,y0+h/2*l2,z0+h/2*m2)
k4=f1(x0+h*k3,y0+h*l3)
l4=f2(x0+h*k3,y0+h*l3,z0+h*m3)
m4=f3(x0+h*k3,y0+h*l3,z0+h*m3)

h=h/2.0

k21=f1(x0,y0)
l21=f2(x0,y0,z0)
m21=f3(x0,y0,z0)
k22=f1(x0+h/2*k21,y0+h/2*l21)
l22=f2(x0+h/2*k21,y0+h/2*l21,z0+h/2*m21)
m22=f3(x0+h/2*k21,y0+h/2*l21,z0+h/2*m21)
k23=f1(x0+h/2*k22,y0+h/2*l22)
l23=f2(x0+h/2*k22,y0+h/2*l22,z0+h/2*m22)
m23=f3(x0+h/2*k22,y0+h/2*l22,z0+h/2*m22)
k24=f1(x0+h*k23,y0+h*l23)
l24=f2(x0+h*k23,y0+h*l23,z0+h*m23)
```

```
m24=f3(x0+h*k23,y0+h*l23,z0+h*m23)

if (abs(k22-k2) > acc) then
goto 100
end if

if (abs(k23-k3) > acc) then
goto 100
end if

if (abs(k24-k4) > acc) then
goto 100
end if

if (abs(l22-l2) > acc) then
goto 100
end if

if (abs(l23-l3) > acc) then
goto 100
end if

if (abs(l24-l4) > acc) then
goto 100
end if

if (abs(m22-m2) > acc) then
goto 100
end if

if (abs(m23-m3) > acc) then
goto 100
end if

if (abs(m24-m4) > acc) then
goto 100
end if


x=x0+1/6.0*(k1+2*k2+2*k3+k4)*2*h
y=y0+1/6.0*(l1+2*l2+2*l3+l4)*2*h
z=z0+1/6.0*(m1+2*m2+2*m3+m4)*2*h
t=t+2*h
x0=x
y0=y
z0=z
end do
call cpu_time(t2)
```

```
write(*,*)"THE RESULT HAS BEEN WRITEN IN THE TXT FILE"
write(*,*)t2-t1
stop
end

real function f1(x,y)
real x,y
real :: a=10.0
f1=a*(y-x)
return
end

real function f2(x,y,z)
real x,y,z
real :: c=28
f2=c*x-y-x*z
return
end

real function f3(x,y,z)
real x,y,z
real :: b=8.0/3.0
f3=-b*z+x*y
return
end

import matplotlib.pyplot as plt

from fft import fft, fft_power, ifft
from numpy import array, real
import math
import time



file = open("wh_butterfly.txt", 'r')
lines = file.readlines()
x = []
y = []
z = []
for line in lines:
    if line != '\n':
        words = line.split()
        ix, iy, iz = [float(s) for s in words]
        x.append( ix )
        y.append( iy )
        z.append( iz )
file.close()
```

```
#fft part
N = len(x)
log2N = math.log(N, 2)
if log2N - int(log2N) > 0.0 :
    print 'Padding with zeros!'
    pads = [0.0] * (pow(2, int(log2N)+1) - N)
    x = x + pads
    y = y + pads
    z = z + pads
    N = len(x)
    print len(y)

X = fft(x)
Y = fft(y)
Z = fft(z)
t = []
for i in range(len(X)):
    t.append(i)
#Yre = [math.sqrt(Y[i].real**2+Y[i].imag**2) for i in xrange(len(Y))]
#powery = fft_power(Y)
#powerx = array([ float(i) for i in xrange(len(powery)) ] )

ax1 = plt.subplot(1, 1, 1)
#p1, = plt.plot( x, y )
#p2, = plt.plot( x, ysmoothedreal )
p1, = plt.plot( t, X )
ax1.legend( [p1], ['Z fft'] )

#ax2 = plt.subplot(3, 1, 2)
#p3, = plt.plot( t, X )
#ax2.legend( [p3], ["X fft"] )

#ax3 = plt.subplot(3, 1, 2)
#p4, = plt.plot( t, Z )
#ax3.legend( [p4], ["Z fft"] )
plt.show()
```