PHYS580 Homework 2

Yicheng Feng
PUID: 0030193826

September 21, 2019

Workflow: I use the Linux system, and code C++ in terminals. For visualization, I use the C++ package – ROOT (made by CERN) to make plots. At the end, the reports are written in LaTeX.

The codes for this lab are written as the following files:

- `runge_kutta.h` and `runge_kutta.cxx` for the class `RungeKutta` to solve general ordinary differential equation sets.
- `pendulum.h` and `pendulum.cxx` for the class `Pendulum` to calculate the motion of physical pendulums.
- `anharmonics.h` and `anharmonics.cxx` for the class `Anharmonics` to calculate the motion anharmonic oscillators.
- `hw2.cxx` for the `main` function to make plots.

To each problem of this lab report, I will attach the relevant parts of the code. If you want to check the validation of my code, you need to download the whole code from the link `https://github.com/YichengFeng/phys580/tree/master/hw2`.

(1) For the driven, nonlinear pendulum, investigate how the Poincare section depends on the strength of the linear dissipation term. Use the same physics parameters as the calculation in Fig. 3.6 of the Giordano-Nakanishi book. First set $F_D = 0.5$, $q = 0.5$, which puts you in the periodic regime, and then vary q to see how the Poincare section changes. Next, repeat the same from a starting point $F_D = 1.2$, $q = 0.5$ in the chaotic regime. In both cases, keep all other physics parameters fixed as you vary q.

Physics explanation:

Same as Fig. 3.6 of the Giordano-Nakanishi book, we set the parameters: length $L = 9.8$ m, gravity $g = 9.8$ m s^{-2}, the initial angle $\theta_0 = 0.2$ rad, the initial angular velocity $\omega_0 = 0$, and the time step $\Delta t = 0.04$ s. For the driving force, the angular frequency is $\Omega_D = 2/3$ and the initial phase is 0. For Poincare section, the section frequency is also $\Omega_S = 2/3$, and the phase cut is also 0. To make the error smaller in the numerical calculation, we use the Runge-Kutta 4$^{\text{th}}$-order approximation.

When $F_D = 0.5$, we scan the q value from 0 to 1 (Fig. 1). We can see several regimes and their representative points: (a) normal period regime, $q = 0.50$; (b) period doubling regime, $q = 0.12$; (c) chaotic regime, $q = 0.05$. The results of the Poincare section of the representative points are shown in Fig. 2. We can see: (a) in the normal period regime, when stable, there is only one section dot in the each phase plot (θ-ω, ω-energy); (b) in the period doubling regime, the representative point is a 3-period point, so when stable, there are three section dots in the phase plots. (c) in the chaotic regime, the section gives dots distributing widely, but not the full phase space.

When $F_D = 1.2$, we scan the q value from 0 to 1 (Fig. 3). We can see several regimes and their representative points: (a) normal period regime, $q = 0.70$ (one section dot when stable); (b) period doubling regime, $q = 0.37$ (2-period, two section dots when stable); (c) chaotic regime, $q = 0.10$ (section dots distributing widely, but not the full phase space). The results of the Poincare section of the representative points are shown in Fig. 4.

We can make some observations. For large q, the maximum energy is smaller. For large F_D, it needs larger q to reach the normal period regime and there seems to be more period doubling regimes and larger chaotic regimes.

Plots:

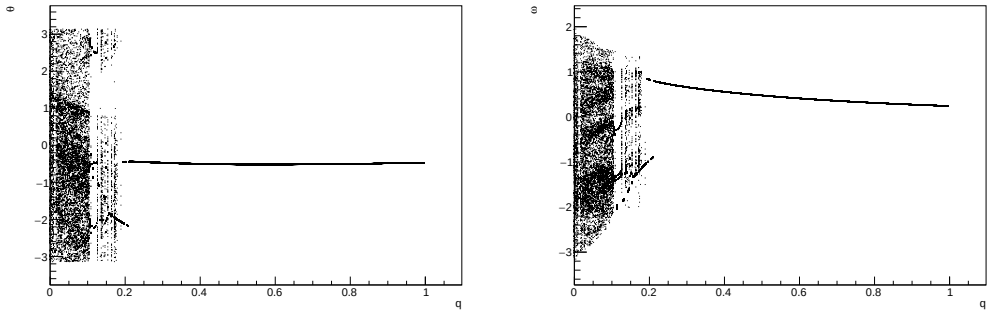

Figure 1: The bifurcation diagram for the pendulum in the vicinity of $q = 0$ to 1, when $F_D = 0.5$.

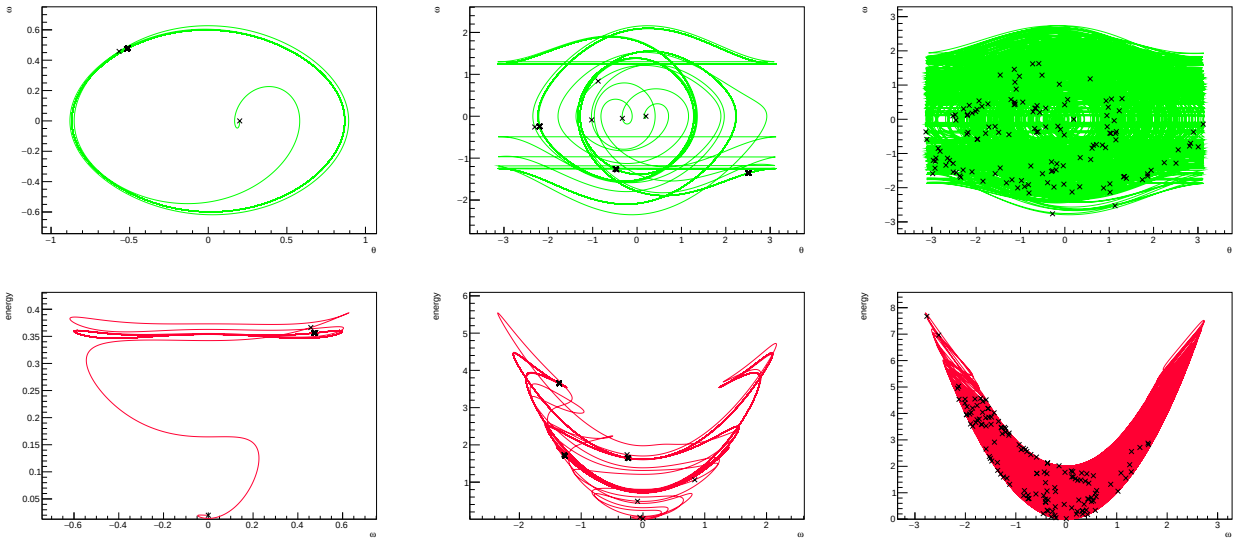

Figure 2: (a) **Normal period regime** (left column): Poincare section at $q = 0.50$ and $F_D = 0.5$. (b) **Period doubling regime** (3-period) (middle column): Poincare section at $q = 0.12$ and $F_D = 0.5$. (c) **Chaotic regime** (right column): Poincare section at $q = 0.05$ and $F_D = 0.5$.

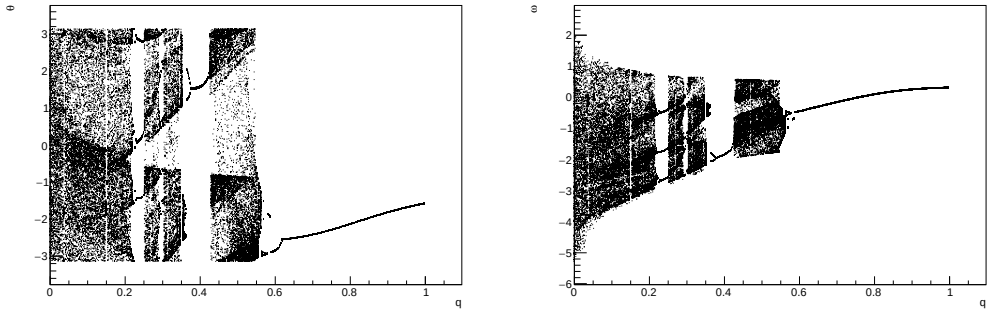

Figure 3: The bifurcation diagram for the pendulum in the vicinity of $q = 0$ to 1, when $F_D = 1.2$.

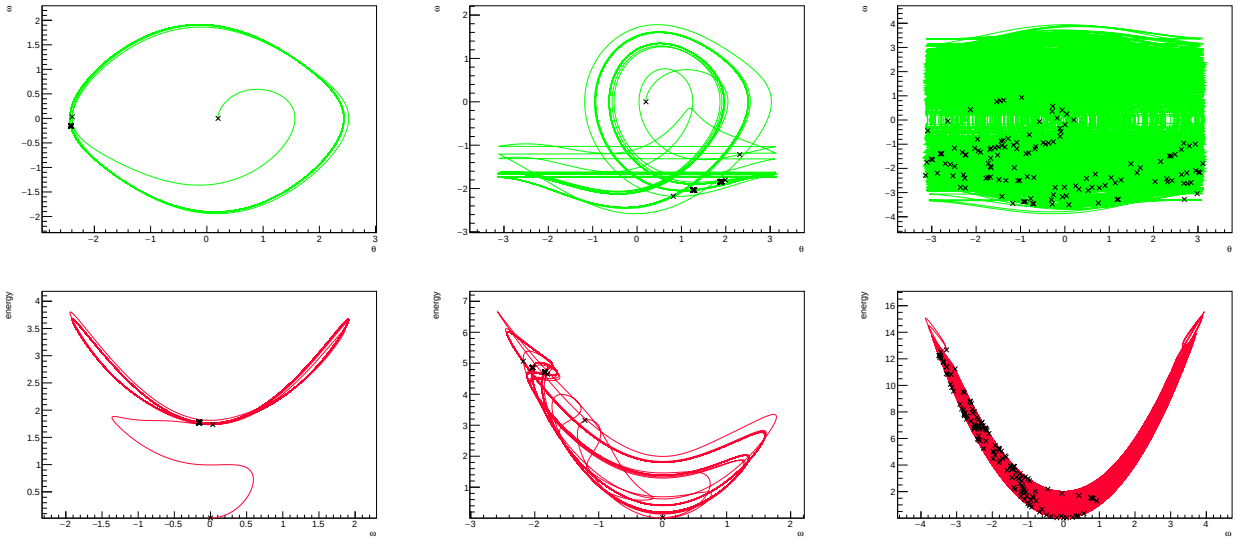

Figure 4: (a) **Normal period regime** (left column): Poincare section at $q = 0.70$ and $F_D = 0.5$. (b) **Period doubling regime** (3-period) (middle column): Poincare section at $q = 0.37$ and $F_D = 0.5$. (c) **Chaotic regime** (right column): Poincare section at $q = 0.10$ and $F_D = 0.5$.

Relevant code: For the Runge-Kutta 4$^{\text{th}}$-order approximation

```
//------------------------------------------------------------------------//

void RungeKutta::cal_rk4() {
        if(!check() || !clear()) {
                cout << "WARNING: check() not passed, no calculation!" << endl;
                return;
        }
        //cout << "check() passed" << endl;

        _n_stps = 0;

        double t = _t_start;
        vector<double> x = _x_start;

        vector<double> F1(_n_eqns);
        vector<double> F2(_n_eqns);
        vector<double> F3(_n_eqns);
        vector<double> F4(_n_eqns);

        vector<double> x1(_n_eqns);
        vector<double> x2(_n_eqns);
        vector<double> x3(_n_eqns);
        vector<double> x4(_n_eqns);

        double t1, t2, t3, t4;

        int nfv = _fv.size(); // nfv = _n_eqns
        while(!_stop(t, x)) {
                // Fill into output
                _t.push_back(t);
                for(int ifv=0; ifv<nfv; ifv++) {
                        _x[ifv].push_back(x[ifv]);
                }
```

```
                _n_stps ++;

                // F1
                t1 = t;
                for(int ifv=0; ifv<nfv; ifv++) {
                        x1[ifv] = x[ifv];
                }
                for(int ifv=0; ifv<nfv; ifv++) {
                        F1[ifv] = _fv[ifv](t1, x1);
                }
                // F2
                t2 = t + 0.5*_dt;
                for(int ifv=0; ifv<nfv; ifv++) {
                        x2[ifv] = x[ifv] + F1[ifv]*0.5*_dt;
                }
                for(int ifv=0; ifv<nfv; ifv++) {
                        F2[ifv] = _fv[ifv](t2, x2);
                }
                // F3
                t3 = t + 0.5*_dt;
                for(int ifv=0; ifv<nfv; ifv++) {
                        x3[ifv] = x[ifv] + F2[ifv]*0.5*_dt;
                }
                for(int ifv=0; ifv<nfv; ifv++) {
                        F3[ifv] = _fv[ifv](t3, x3);
                }
                // F4
                t4 = t + _dt;
                for(int ifv=0; ifv<nfv; ifv++) {
                        x4[ifv] = x[ifv] + F3[ifv]*_dt;
                }
                for(int ifv=0; ifv<nfv; ifv++) {
                        F4[ifv] = _fv[ifv](t4, x4);
                }

                t += _dt;
                for(int ifv=0; ifv<nfv; ifv++) {
                        x[ifv] += (F1[ifv]/6 + F2[ifv]/3 + F3[ifv]/3 + F4[ifv]/6)*_dt;
                }
        }

        _t.push_back(t);
        for(int ifv=0; ifv<nfv; ifv++) {
                _x[ifv].push_back(x[ifv]);
        }
        _n_stps ++;
}

//------------------------------------------------------------------------//
```

For the ODE set of the physical pendulum

```
//------------------------------------------------------------------------//

// x[0]: theta; x[1]: omega
double Pendulum::f_theta(double t, const vector<double> &x) {
        // dtheta/dt = omega
        return x[1];
}

//------------------------------------------------------------------------//
```

```
10
11 double Pendulum::f_omega_physics(double t, const vector<double> &x) {
12         return -_g/_l*sin(x[0]) - _q*x[1] + _F*sin(_O*t+_P);
13 }
14
15 //------------------------------------------------------------------------//
16
17 double Pendulum::f_energy_physics(double t, const vector<double> &x) {
18         // 2E/m/l^2
19         return _g/_l*(1-cos(x[0]))+x[1]*x[1];
20 }
21
22 //------------------------------------------------------------------------//
```

For the solution of the ODE set and Poincare section:

```
1  //------------------------------------------------------------------------//
2
3  void chao_plot(double f_d, double theta_start, double omega_start, double psphi, double psomega) {
4
5          Pendulum pd;
6          pd.set_mode(1); // physics pendulum
7          pd.set_alg(4);
8          //pd.set_theta_start(0.2);
9          //pd.set_omega_start(0);
10         pd.set_theta_start(theta_start);
11         pd.set_omega_start(omega_start);
12         pd.set_q(0.5);
13         pd.set_F(f_d);
14         pd.set_O(2./3);
15         pd.set_dt(0.04);
16         pd.set_n_periods(400);
17
18         pd.cal();
19
20         // solution of the ODE set
21         vector<double> t = pd.get_t();
22         vector< vector<double> > x = pd.get_x();
23         vector<double> energy = pd.get_energy();
24         int n_stps = pd.get_n_stps();
25
26         theta_to_twopi(x[0]);
27
28         // fixed poincare section
29         vector<double> ps_f_d;
30         vector<double> ps_theta;
31         vector<double> ps_omega;
32         vector<double> ps_energy;
33         int ps_n_pts = 0;
34
35         //double psphi = 0;
36         //double psomega = 2./3;
37         double dt = pd.get_dt();
38
39         for(int i_stps=0; i_stps<n_stps; i_stps++) {
40                 if(fabs(psomega*t[i_stps]-psphi-int((psomega*t[i_stps]-psphi)/2./M_PI)*2*M_PI)
41                 > 0.5*psomega*dt*0.999999) {
42                         continue;
43                 }
44                 ps_f_d.push_back(f_d);
45                 ps_theta.push_back(x[0][i_stps]);
```

```
                ps_omega.push_back(x[1][i_stps]);
                ps_energy.push_back(energy[i_stps]);
                ps_n_pts ++;
        }

        // ... plotting code ...
}

//------------------------------------------------------------------------//
```

For the q scan (bifurcation diagram)

```
//------------------------------------------------------------------------//

void poincare_section_scan(double f_d) {

        TString str_f_d = Form("fd_%.3d", int(f_d*100));

        double omega_d = 2./3;
        //double f_d = 1.30;
        double dt = 0.04;
        double T_d = 2*M_PI/omega_d;
        double psphi = 0;
        double psomega = 2./3;
        double q = 0.5;

        vector<double> ps_q;
        vector<double> ps_theta;
        vector<double> ps_omega;
        vector<double> ps_energy;
        int ps_n_pts = 0;

        double q_min = 0;
        double q_max = 1;
        int n_q = 400;

        for(int i=0; i<n_q; i++) {

                q = q_min+(q_max-q_min)/n_q*i;

                Pendulum pd;
                pd.set_mode(1); // physics pendulum
                pd.set_alg(4);
                pd.set_theta_start(0.2);
                pd.set_omega_start(0);
                pd.set_q(q);
                pd.set_F(f_d);
                pd.set_O(omega_d);
                pd.set_dt(dt);
                pd.set_n_periods(400/omega_d);

                pd.cal();

                vector<double> t = pd.get_t();
                vector< vector<double> > x = pd.get_x();
                vector<double> energy = pd.get_energy();
                int n_stps = pd.get_n_stps();

                theta_to_twopi(x[0]);

                for(int i_stps=300/omega_d/dt; i_stps<n_stps; i_stps++) {
```

```
50                          if(fabs(psomega*t[i_stps]-psphi-int((psomega*t[i_stps]-psphi)/2./M_PI)*2*M_PI)
51                                  continue;
52                          }
53                          ps_q.push_back(q);
54                          ps_theta.push_back(x[0][i_stps]);
55                          ps_omega.push_back(x[1][i_stps]);
56                          ps_energy.push_back(energy[i_stps]);
57                          ps_n_pts ++;
58                  }
59          }
60
61          // ... plot code ...
62  }
63  //------------------------------------------------------------------------//
```

(2) [*Problem 3.4 (p.53)*] For simple harmonic motion, the general form for the equation of motion is

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -k\mathtt{sgn}(x)|x|^\alpha, \tag{1}$$

with $\alpha = 1$. This has the same form as (3.2), although the variables have different names. Begin by writting a program that uses the Euler-Cromer method to solve for x as a function of time according to Eq. 3, with $\alpha = 1$ (for convenience, you can take $k = 1$). The subroutine described in this section can be modified to accomplish this. Show that the period of the oscillations is independent of the amplitude of the motion. This is a key feature of simple harmonics motion. Then extend your program to treat the cas $\alpha = 3$. This is an example of an anharmonics oscillator. Calculate the period of the oscillations for several different amplitudes (amplitudes in the range 0.2 to 1 are good choices), and show that the period of the motion now depends on the amplitude. Given an intuitive argument to explain why the period becomes longer as the amplitude is decreased.

Physics explanation:

Similar as before, we expand this second order differential equation into an ODE set of two first order differential equations:

$$\begin{cases} \dfrac{\mathrm{d}x}{\mathrm{d}t} = v \\ \dfrac{\mathrm{d}v}{\mathrm{d}t} = -k\mathtt{Sgn}(x)|x|^\alpha \end{cases}. \tag{2}$$

Then, we use the Euler-Cromer approxition to solve this ODE set numerically.

To study the initial conditions' effect to the solution, we set three sets of amplitudes: (a) $x_m = 0.2$; (b) $x_m = 0.6$; (c) $x_m = 1.0$. We set the parameter $k = 1$ and study the three conditions in both cases $\alpha = 1$ and $\alpha = 3$.

The results are shown in Fig. 5. The left pad is the case $\alpha = 1$. We can see from both the curves and the legends, that the periods are the same for whatever amplitude we set.

The right pad is the case $\alpha = 3$. However, we can see from both the curves and the legends, that the periods are different for the different amplitudes we set. The larger amplitude we set, the smaller period it will have. This could be understood in the following way.

We call the invariant period T of $\alpha = 1$. It is easy to know that $T = \frac{2\pi}{\sqrt{k}}$. For $\alpha > 1$, we have

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -k\mathtt{Sgn}(x)|x|^\alpha = -k'(x)x, \tag{3}$$

where the $k'(x) = k|x|^{\alpha-1}$ could be estimate by $k'(x) \approx kx_m^{\alpha-1}$. Then the period of the anharmonics can be estimated as

$$T' \approx \frac{2\pi}{\sqrt{k'}} \approx \frac{2\pi}{\sqrt{k}} x_m^{\frac{1-\alpha}{2}}, \tag{4}$$

where $\frac{1-\alpha}{2} < 0$, so T' decreases along x_m – the smaller x_m can give larger T'.

In an intuitive way, we can say the "force" in anharmonics ($\alpha = 3$) increases faster than the harmonics ($\alpha = 1$) along x, so it goes faster in the higher x range than the harmonics. Since the harmonics has an invariant period, the anharmonics must have an smaller period with larger x_m.

Plots:

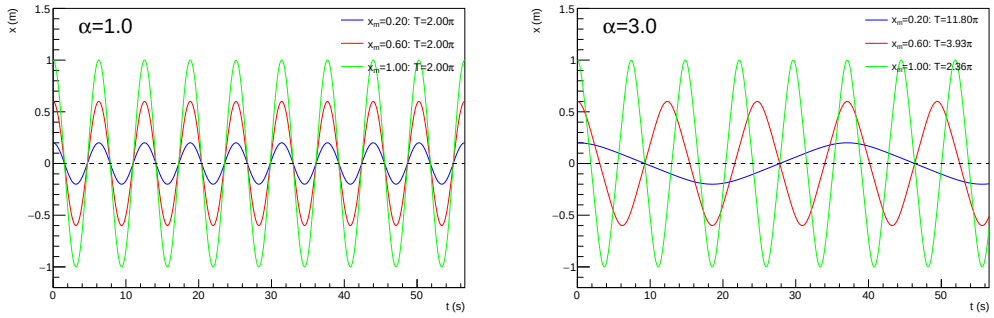

Figure 5: The solution of Eq. 3 with different amplitudes ($x_m = 0.2, 0.6, 1.0$). The left pad has $\alpha = 1$ and the right pad has $\alpha = 3$.

Relevant code:

For the Euler-Cromer approximation

```
//------------------------------------------------------------------------------//

void RungeKutta::cal_Euler_Cromer() {
        if(!check() || !clear()) {
                cout << "WARNING: check() not passed, no calculation!" << endl;
                return;
        }
        //cout << "check() passed" << endl;

        _n_stps = 0;

        double t = _t_start;
        vector<double> x = _x_start;

        vector<double> F1(_n_eqns);

        int nfv = _fv.size(); // nfv = _n_eqns
        while(!_stop(t, x)) {
                // Fill into output
                _t.push_back(t);
                for(int ifv=0; ifv<nfv; ifv++) {
                        _x[ifv].push_back(x[ifv]);
                }
                _n_stps ++;

                // F1
                // update x right after each calculation
                //for(int ifv=0; ifv<nfv; ifv++) {
                for(int ifv=nfv-1; ifv>=0; ifv--) {
                        F1[ifv] = _fv[ifv](t, x);
                        x[ifv] += F1[ifv]*_dt;
                }
                t += _dt;
        }

        _t.push_back(t);
        for(int ifv=0; ifv<nfv; ifv++) {
                _x[ifv].push_back(x[ifv]);
        }
```

```
        _n_stps ++;
}

//------------------------------------------------------------------------//
```

For the ODE set of the anharmonics

```
//------------------------------------------------------------------------//

double Anharmonics::f_theta(double t, const vector<double> &x) {
        return x[1];
}

//------------------------------------------------------------------------//

double Anharmonics::f_omega(double t, const vector<double> &x) {
        int sign = 0;
        if(x[0]>0) sign = 1;
        if(x[0]<0) sign = -1;
        return -_k*sign*pow(fabs(x[0]), _alpha);
}

//------------------------------------------------------------------------//
```

(3) [*Problem 3.5 (p.54)*] For the anharmonics oscillator of Eq. 3 in the previous exercise, it is possible to analytically obtain the period of oscillation for general values of α in terms of certain special functions. Do this, and describe how the relationship between the period and the amplitude depends on the value of α. Can you give a physical interpretation to the finding? Also check how well the periods computed in Problem (2) agree with the analytic results, and discuss whether the difference you find is within the expected accuracy of the numerical calculation.

Analytical approach to the period of anharmonics:

We start from Eq. 3 and focus on the range $x > 0$

$$\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -kx^\alpha \quad \Rightarrow \quad \frac{\mathrm{d}x}{\mathrm{d}t}\frac{\mathrm{d}^2x}{\mathrm{d}t^2} = -kx^\alpha\frac{\mathrm{d}x}{\mathrm{d}t} \quad \Rightarrow \quad \frac{1}{2}\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)^2 = -\frac{k}{\alpha+1}\frac{\mathrm{d}}{\mathrm{d}t}\left(x^{\alpha+1}\right) \tag{5}$$

We integrate over t the both sides of the rightmost equation above. We set the initial condition $x(0) = x_m$ and $v(0) = 0$, and have

$$\frac{1}{2}\left(\frac{\mathrm{d}x}{\mathrm{d}t}\right)^2 = \frac{k}{\alpha+1}\left(x_m^{\alpha+1} - x^{\alpha+1}\right) \quad \Rightarrow \quad \frac{\mathrm{d}x}{\mathrm{d}t} = \sqrt{\frac{2k}{\alpha+1}\left(x_m^{\alpha+1} - x^{\alpha+1}\right)}, \tag{6}$$

$$\mathrm{d}t = \sqrt{\frac{\alpha+1}{2k}}\left(x_m^{\alpha+1} - x^{\alpha+1}\right)^{-1/2}\mathrm{d}x = \sqrt{\frac{\alpha+1}{2k}}x_m^{\frac{1-\alpha}{2}}\left(1-\xi^{\alpha+1}\right)^{-1/2}\mathrm{d}\xi, \tag{7}$$

where $\xi = x/x_m$. For a quarter period, x goes from 0 to x_m,

$$T/4 = \int_{x=0}^{x=x_m} dt = \sqrt{\frac{\alpha+1}{2k}}x_m^{\frac{1-\alpha}{2}}\int_0^1 \left(1-\xi^{\alpha+1}\right)^{-1/2}\mathrm{d}\xi = \sqrt{\frac{\alpha+1}{2k}}x_m^{\frac{1-\alpha}{2}}\sqrt{\pi}\frac{\Gamma\left(1+\frac{1}{\alpha+1}\right)}{\Gamma\left(\frac{1}{2}+\frac{1}{\alpha+1}\right)}, \tag{8}$$

the last equation comes from integration I_n provided in Piazza. Then, the period of the anharmonics oscillator is

$$T = 4\sqrt{\frac{\pi}{2k}(\alpha+1)}x_m^{\frac{1-\alpha}{2}}\frac{\Gamma\left(1+\frac{1}{\alpha+1}\right)}{\Gamma\left(\frac{1}{2}+\frac{1}{\alpha+1}\right)}. \tag{9}$$

Physics explanation:

From Eq. 9, the relationship between T and x_m is $T \propto x_m^{\frac{1-\alpha}{2}}$. For $0 < \alpha < 1$, the period T increases along the amplitude x_m; for $\alpha = 1$, the period T is independent from the amplitude x_m; for $\alpha > 1$, the period T decreases along the amplitude x_m.

Check with problem (2):

We set $k = 1$. When $\alpha = 1$, $T = 4\sqrt{\frac{\pi}{2}(1+1)}\Gamma(3/2)/\Gamma(1) = 4\sqrt{\pi}\sqrt{\pi}/2/1 = 2\pi$, which is independent from x_m and consistent with the numerical calculaiton in (2). The previous number for the period with $\alpha = 1$ is 1.9999667π. The relative difference is about 1.7×10^{-5}.

When $\alpha = 3$, $T = 4\sqrt{\frac{\pi}{2}(1+3)}x_m^{-1}\Gamma(5/4)/\Gamma(3/4) \approx 7.4163x_m^{-1}$.

amplitude x_m	0.2	0.6	1.0
numerical period T_n from (2)	11.803379π	3.9343891π	2.3605486π
analytical period T from (3)	11.803406π	3.9344687π	2.3606812π
relative error $\lvert T_n - T\rvert/T$	2.3×10^{-6}	2.0×10^{-5}	5.6×10^{-5}

The difference is small and should be within the expected accuracy of the numerical calculation.

(4) [*Problem 3.13 (p.65)*] Write a program to calculate and compare the behavior of two, nearly identical pendulums. Use it to calculate the divergence of two nearby trajectories in the chaotic regime, as in Figure 3.7, and make a qualitative estimate of the corresponding Lyapunov exponent from the slope of a plot of $\log(\Delta\theta)$ as a function of t.

Physics explanation:

The key property of the chaotic system is that the difference at the initial state will increase along time exponentially.

We follow the setting in the textbook Figure 3.6 and 3.7: length $L = 9.8$ m, gravity $g = 9.8$ m s^{-2}, the initial angle $\theta_0 = 0.2$ rad, the initial angular velocity $\omega_0 = 0$, and the time step $\Delta t = 0.04$ s. For the driving force, the angular frequency is $\Omega_D = 2/3$ and the initial phase is 0. For the other pendulum, we vary the initial condition to $\theta_0 = 0.201$ rad. We use the Runge-Kutta 4$^{\text{th}}$-order approximation.

Figure 6 left pad shows the time evolution of the two pendulums' θ. The right pad show the difference between them $\Delta\theta$ depending on time in the logarithm scale. Qualitatively, we can estimate the corresponding Lyapunov exponent from the slope of the trend (the red dashed line), whose slope is $k = 0.170$.

Plots:

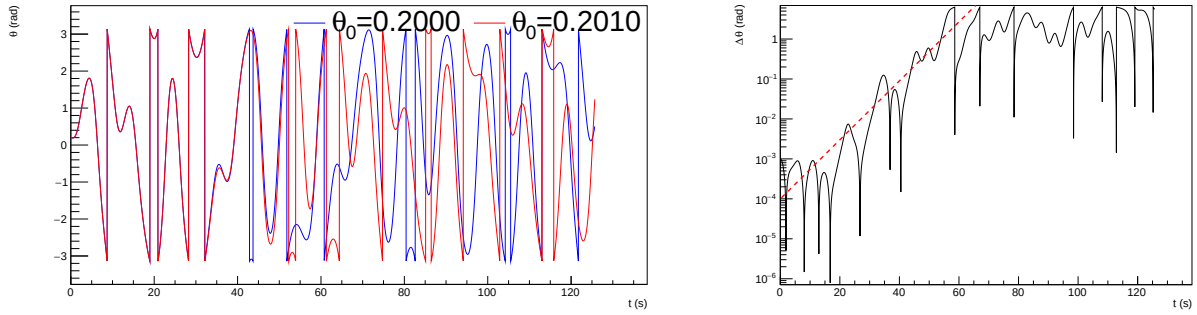

Figure 6: The time evolution of the two pendulums (left). The θ difference between the two identical pendulum (right pad) with slight different initial condition $\theta_0 = 0.2$ rad vs $\theta_0 = 0.201$ rad.

Relevant code:

The relevant code of the Runge-Kutta 4$^{\text{th}}$-order approximation and the ODE set of the physical pendulum are already listed in problem (1).

For the difference

```
1  //------------------------------------------------------------------------//
2
3  void chao_delta_start() {
4
5          double dtheta0 = 0.001;
6          double f_d = 1.2;
7          double theta_start1 = 0.2;
8          double theta_start2 = theta_start1 + dtheta0;
9          double omega_start1 = 0;
10         double omega_start2 = 0;
11
12         Pendulum pd1;
13         pd1.set_mode(1); // physics pendulum
14         pd1.set_alg(4);
15         pd1.set_theta_start(theta_start1);
```

```
        pd1.set_omega_start(omega_start1);
        pd1.set_q(0.5);
        pd1.set_F(f_d);
        pd1.set_O(2./3);
        pd1.set_dt(0.02);
        pd1.set_n_periods(20);
        pd1.cal();

        vector<double> t1 = pd1.get_t();
        vector< vector<double> > x1 = pd1.get_x();
        int n_stps1 = pd1.get_n_stps();

        Pendulum pd2;
        pd2.set_mode(1); // physics pendulum
        pd2.set_alg(4);
        pd2.set_theta_start(theta_start2);
        pd2.set_omega_start(omega_start2);
        pd2.set_q(0.5);
        pd2.set_F(f_d);
        pd2.set_O(2./3);
        pd2.set_dt(0.02);
        pd2.set_n_periods(20);
        pd2.cal();

        vector<double> t2 = pd2.get_t();
        vector< vector<double> > x2 = pd2.get_x();
        int n_stps2 = pd2.get_n_stps();

        vector<double> dtheta;
        for(int i=0; i<t1.size(); i++) {
                double tmp_dtheta = fabs(x2[0][i]-x1[0][i]);
                while(tmp_dtheta > 2*M_PI) tmp_dtheta -= 2*M_PI;
                dtheta.push_back(tmp_dtheta);
        }
}

//-----------------------------------------------------------------------//
```

(5) [*Problem 3.20 (p.70)*] Calculate the bifurcation diagram for the pendulum in the vicinty of $F_D = 1.35$ to 1.5. Make a magnified plot of the diagram (as compared to Figure 3.11) and obtain an estimate of the Feigenbaum δ parameter.

Physics explanation:

We follow the setting in the textbook Figure 3.6: length $L = 9.8$ m, gravity $g = 9.8$ m s^{-2}, the initial angle $\theta_0 = 0.2$ rad, the initial angular velocity $\omega_0 = 0$, and the time step $\Delta t = 0.04$ s. For the driving force, the angular frequency is $\Omega_D = 2/3$ and the initial phase is 0. We use the Runge-Kutta 4$^{\text{th}}$-order approximation.

To remove the effect of the initial condition, we let the pendulum oscillate for 350 driving force periods, and then do the Poincare section for 50 driving force periods. The results are shown in Fig. 7. The left pad shows the Poincare section of θ depending on F_D, and the right pad shows ω depending on F_D.

According to the textbook, we define F_n to be the value of the driving force at which the transition to period-2^n behavior takes place. From Fig. 7, we can find the values: $F_1 = 1.424$, $F_2 = 1.459$, and $F_3 = 1.477$. Then we can estimate the value of Feigenbaum δ_2

$$\delta_2 = \frac{F_2 - F_1}{F_3 - F_2} = \frac{1.459 - 1.424}{1.477 - 1.459} = 1.944 \tag{10}$$

From the textbook, when n goes infinity, the limit of δ_n goes $\delta = 4.669$, which is different from our δ_2. This is because the order we have is only $n = 2$, too small. When we can get higher order F_n, we may find the δ_n closer to the limit.

Plots:

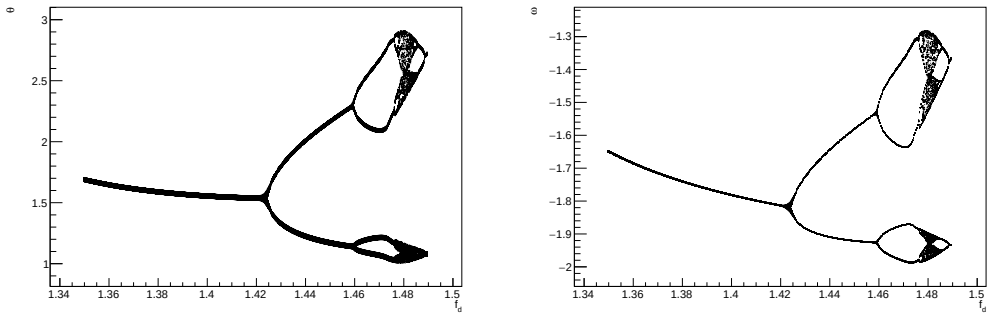

Figure 7: The bifurcation diagram for the pendulum in the vicinity of $F_D = 1.35$ to 1.50.

Relevant code:

The relevant code of the Runge-Kutta 4$^{\text{th}}$-order approximation and the ODE set of the physical pendulum are already listed in problem (1).

For the bifurcation diagram

```
1 //--------------------------------------------------------------------------//
2
3 void poincare_section_scan_fd(double psphi, double psomega) {
4
5         TString str_psphi = Form("psphi_%.3d", int(psphi*100));
6         TString str_psomega = Form("psomega_%.3d", int(psomega*100));
7         TString str_q = Form("q_050");
8
9         double omega_d = 2./3;
```

```
        double f_d = 1.30;
        double dt = 0.04;
        double T_d = 2*M_PI/omega_d;

        vector<double> ps_f_d;
        vector<double> ps_theta;
        vector<double> ps_omega;
        vector<double> ps_energy;
        int ps_n_pts = 0;

        double f_d_min = 1.35;
        double f_d_max = 1.49;
        int n_f_d = 400;

        for(int i=0; i<n_f_d; i++) {

                f_d = f_d_min+(f_d_max-f_d_min)/n_f_d*i;

                Pendulum pd;
                pd.set_mode(1); // physics pendulum
                pd.set_alg(4);
                pd.set_theta_start(0.2);
                pd.set_omega_start(0);
                pd.set_q(0.5);
                pd.set_F(f_d);
                pd.set_O(omega_d);
                pd.set_dt(dt);
                pd.set_n_periods(400/omega_d);

                pd.cal();

                vector<double> t = pd.get_t();
                vector< vector<double> > x = pd.get_x();
                vector<double> energy = pd.get_energy();
                int n_stps = pd.get_n_stps();

                theta_to_twopi(x[0]);

                for(int i_stps=350/omega_d/dt; i_stps<n_stps; i_stps++) {
                        if(fabs(psomega*t[i_stps]-psphi-int((psomega*t[i_stps]-psphi)/2./M_PI)*2*M_PI)
                                continue;
                        }
                        ps_f_d.push_back(f_d);
                        ps_theta.push_back(x[0][i_stps]);
                        ps_omega.push_back(x[1][i_stps]);
                        ps_energy.push_back(energy[i_stps]);
                        ps_n_pts ++;
                }
        }

        // ... plot code ...
}
//------------------------------------------------------------------------------//
```