PHYS580 Lab02 Report

Yicheng Feng
PUID: 0030193826

September 4, 2019

Before my solutions to the lab activities, I want to set up my typical **workflow**, which will also be used in this course. 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 five files:

- `runge_kutta.h` and `runge_kutta.cxx` for the class `RungeKutta` to solve general ordinary differential equation sets.
- `projectile.h` and `projectile.cxx` for the class `Projectile` to calculate the cannon shell projectiles with drags of various air model.
- `lab2.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/lab2`.

(1) First, for a cannon shell projectile, take a fixed angle of projection of 45°, and an initial velocity of 700 m/s, and graphically compare trajectories for the following different models: (a) gravity only, no air drag, (b) air drag with constant air density, (c) air drag with the isothermal model of air density, and (d) air drag with the adiabatic model, using $\gamma = 1.4$. Set $B_2/m = 4 \times 10^{-5}$ (in 1/m units) at ground level. In the adiabatic model, $|F_{\rm drag}| = B_2(1 - ay/T_{\rm grd})^{1/(\gamma-1)}v^2$, where a is the rate of temperature decrease per rise from sea level. Use $a = 6.5 \times 10^{-3}$ K/m, and $T_{\rm grd} = 293$ K. [Can you derive/justify this form of F drag ?] Give some arguments for why the comparative shapes of the 4 trajectories, and the corresponding ranges, may be expected.

Plots:

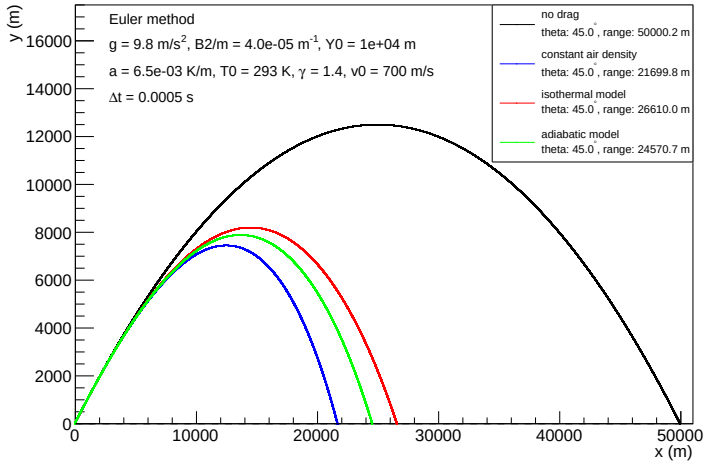

Figure 1: Euler approximation for the cannon shell projectile with different air drag models. The shooting angle θ is the angle between the initial velocity and the ground. Here, the input $\Delta t = 0.0005$ s.

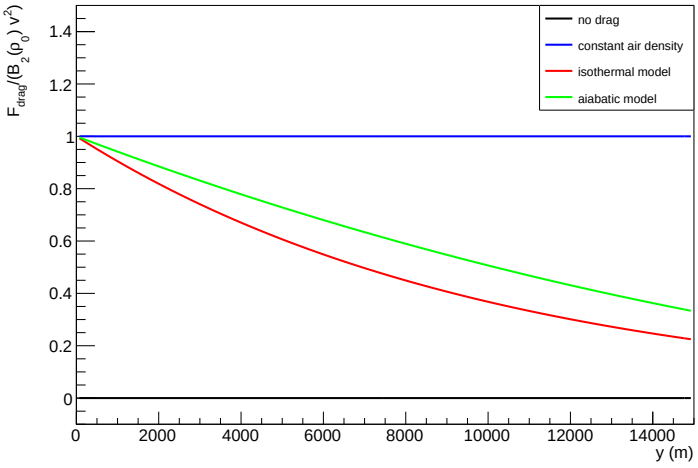

Figure 2: The normalized drag of various models depending on the height y

Physics explanation:

There are four models of air drag mentioned in this problem: (a) gravity only, no air drag, (b) air drag with constant air density, (c) air drag with the isothermal model of air density, and (d) air drag

with the adiabatic model. To make it clear, here lists the formulas of them:

$$\begin{aligned}
&\text{(a) } \vec{F}_{\text{drag}} = 0, \\
&\text{(b) } \vec{F}_{\text{drag}} = -B_2 v^2 \hat{v}, \\
&\text{(c) } \vec{F}_{\text{drag}} = -B_2 e^{-mgy/(kT_0)} v^2 \hat{v} = -B_2 e^{-y/Y_0} v^2 \hat{v}, \\
&\text{(d) } \vec{F}_{\text{drag}} = -B_2 (1 - ay/T_0)^{1/(\gamma-1)} v^2 \hat{v},
\end{aligned} \tag{1}$$

where B_2 is a just constant regarding the air density on the ground (ρ_0). All the variables with subscript "0" are measured on the ground ($y = 0$).

The air density dependence on the height (y) is written explicitly right after B_2. In the starter Matlab program for Lab2, Y_0 is approximated as 1×10^4 m. Here, we can see this number is reasonable by the following calculation. The average mass of air molecule is

$$m = 28.96u = 28.96 \times 1.66 \times 10^{-27} \text{ kg} = 4.81 \times 10^{-26} \text{ kg}. \tag{2}$$

Then, the characteristic length in atmospheric pressure is

$$Y_0 = \frac{kT_0}{mg} = \frac{1.38 \times 10^{-23} \text{ m}^2 \text{ kg s}^{-2} \text{ K}^{-1} \times 293 \text{ K}}{4.81 \times 10^{-26} \text{ kg} \times 9.8 \text{m s}^{-2}} = 8.58 \times 10^3 \text{ m} \approx 1 \times 10^4 \text{ m} \tag{3}$$

The formula of (c) is given by the Boltzmann statistics.

Now, I derive the form of $\vec{F}_{\text{drag}}$ in the adiabatic model. The adiabatic condition gives

$$pV^\gamma = p_0 V_0^\gamma. \tag{4}$$

The particle number N is the same and the number density is N/V, then

$$p \left(\frac{N}{\rho}\right)^\gamma = p_0 \left(\frac{N}{\rho_0}\right)^\gamma \quad \Rightarrow \quad p = p_0 \left(\frac{\rho}{\rho_0}\right)^\gamma. \tag{5}$$

From the basic hydrostatics knowledge, the change of pressure is

$$\frac{\mathrm{d}p}{\mathrm{d}y} = -mg\rho. \tag{6}$$

Combining Eq. 5 and Eq. 6, we can get

$$p_0 \frac{\gamma \rho^{\gamma-1}}{\rho_0^\gamma} \frac{\mathrm{d}\rho}{\mathrm{d}y} = -mg\rho. \tag{7}$$

This equation can be easily solved:

$$\int p_0 \frac{\gamma \rho^{\gamma-2}}{\rho_0^\gamma} \mathrm{d}\rho = -\int mg\rho \mathrm{d}y \quad \Rightarrow \quad \frac{p_0}{\rho_0} \frac{\gamma}{\gamma - 1} \left[\left(\frac{\rho}{\rho_0}\right)^{\gamma-1} - 1 \right] = -mgy. \tag{8}$$

From the ideal gas equation, we have

$$pV = NkT \quad \Rightarrow \quad \frac{p}{\rho} = kT \quad \Rightarrow \quad \frac{p_0}{\rho_0} = kT_0. \tag{9}$$

Then, Eq. 8 can be written as:

$$\frac{\rho}{\rho_0} = \left[1 - \frac{\gamma - 1}{\gamma} \frac{mg}{kT_0} y\right]^{1/(\gamma-1)} = \left(1 - \frac{ay}{T_0}\right)^{1/(\gamma-1)}, \tag{10}$$

where $a = (\gamma - 1)mg/(k\gamma)$. The corresponding drag would be

$$\vec{F}_{\text{drag}} = -B_2 \frac{\rho}{\rho_0} v^2 \hat{v} = -B_2 \left(1 - \frac{ay}{T_0}\right)^{1/(\gamma-1)} v^2 \hat{v}. \tag{11}$$

The case with no drag gives symmetric trajectory **shape** of a quadratic curve, as expected. However, the other three cases give unsymmetric trajectory shapes. the x projection from the initial point to the maximum top is larger than that from the maximum top to the final point. That is because the drag reduces the v_x in the flight. The drag also reduce the total energy of the cannon shell during the flight, so the maximum height is also reduced.

From the Fig. 1, we can see that with the same initial condition the **ranges** of the four models are different: (a) no drag > (c) isothermal model > (d) adiabatic model > (b) constant air density model. It is easy to understand the case (a) without drag can have the biggest range. Because the air density decrease along the height y, the air density is highest on the ground. Then in the case (b) with constant air density, the air density above the ground is all overestimated by that on the ground, so the drag is always overestimated as well. As a result, the case (b) has the shortest range. It is not easy to compare the case (c) and (d), so we plot the normalized drag of various models depending on the height y in Fig. 2. We can see the ranking (b) constant air density > (d) adiabatic model > (c) isothermal model > (a) no drag, which is the inversed ranking of the range and therefore makes sense.

Relevant code:

In the file `runge_kutta.cxx`,

```
void RungeKutta::cal_rk1() {
        if(!check()) {
                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> x1(_n_eqns);

        double t1;

        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);
```

```
                }

                t += _dt;
                for(int ifv=0; ifv<nfv; ifv++) {
                        x[ifv] += F1[ifv]*_dt;
                }
        }

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

In the file **projectile.cxx**,

```
// x[0]: x; x[1]: y; x[2]: vx; x[3]: vy
double Projectile::f_x(double t, const vector<double> &x) {
        // dx/dt = vx
        return x[2];
}

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

double Projectile::f_y(double t, const vector<double> &x) {
        // dy/dt = vy
        return x[3];
}

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

double Projectile::f_vx_no_drag(double t, const vector<double> &x) {
        double drag = 0;
        return -drag*x[2];
}

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

double Projectile::f_vy_no_drag(double t, const vector<double> &x) {
        double drag = 0;
        return -drag*x[3]-_g;
}

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

double Projectile::f_vx_constant_drag(double t, const vector<double> &x) {
        double drag = _B2_m*sqrt(x[2]*x[2]+x[3]*x[3]);
        return -drag*x[2];
}

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

double Projectile::f_vy_constant_drag(double t, const vector<double> &x) {
        double drag = _B2_m*sqrt(x[2]*x[2]+x[3]*x[3]);
        return -drag*x[3]-_g;
}

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

double Projectile::f_vx_isothermal_drag(double t, const vector<double> &x) {
```

```
45          double drag = _B2_m*sqrt(x[2]*x[2]+x[3]*x[3])*exp(-x[1]/_Y0);
46          return -drag*x[2];
47  }
48
49  //------------------------------------------------------------------------//
50
51  double Projectile::f_vy_isothermal_drag(double t, const vector<double> &x) {
52          double drag = _B2_m*sqrt(x[2]*x[2]+x[3]*x[3])*exp(-x[1]/_Y0);
53          return -drag*x[3]-_g;
54  }
55
56  //------------------------------------------------------------------------//
57
58  double Projectile::f_vx_adiabatic_drag(double t, const vector<double> &x) {
59          double drag = _B2_m*sqrt(x[2]*x[2]+x[3]*x[3])*pow(1-_a*x[1]/_T0, 1/(_gamma-1));
60          return -drag*x[2];
61  }
62
63  //------------------------------------------------------------------------//
64
65  double Projectile::f_vy_adiabatic_drag(double t, const vector<double> &x) {
66          double drag = _B2_m*sqrt(x[2]*x[2]+x[3]*x[3])*pow(1-_a*x[1]/_T0, 1/(_gamma-1));
67          return -drag*x[3]-_g;
68  }
69
70  //------------------------------------------------------------------------//
71
72  bool Projectile::stop(double t, const vector<double> &x) {
73          // stop when y<0
74          return x[1]<0;
75  }
76
77  //------------------------------------------------------------------------//
78
79  void Projectile::cal() {
80          if(!check()) return;
81
82          RungeKutta rk(_n_eqns, _dt, _t_start, _c_start);
83          rk.load_f(f_x);
84          rk.load_f(f_y);
85          if(_mode == 0) {
86                  rk.load_f(f_vx_no_drag);
87                  rk.load_f(f_vy_no_drag);
88          } else if(_mode == 1) {
89                  rk.load_f(f_vx_constant_drag);
90                  rk.load_f(f_vy_constant_drag);
91          } else if(_mode == 2) {
92                  rk.load_f(f_vx_isothermal_drag);
93                  rk.load_f(f_vy_isothermal_drag);
94          } else if(_mode == 3) {
95                  rk.load_f(f_vx_adiabatic_drag);
96                  rk.load_f(f_vy_adiabatic_drag);
97          } else {
98                  cout << "ERROR: invalid mode!" << endl;
99                  cout << "0: no drag; 1: constant drag; " << flush;
100                 cout << "2: isothermal drag; 3: adiabatic drag." << endl;
101         }
102
103         rk.set_stop(stop);
104
```

```
105         if(_rk_order == 1) {
106                 rk.cal_rk1();
107         } else if(_rk_order == 2) {
108                 rk.cal_rk2();
109         } else if(_rk_order == 4) {
110                 rk.cal_rk4();
111         } else {
112                 cout << "ERROR: invalid order of RK method!" << endl;
113                 cout << "1: RK1 (Euler); 2: RK2; 4: RK4." << endl;
114         }
115
116         _t = rk.get_t();
117         _x = rk.get_x();
118         _n_stps = rk.get_n_stps();
119 }
```

(2) Second, for each of the cannon shell projectile models you implemented in (1), program some reasonable procedure that i) computes the angle of projection that gives maximal range and ii) obtains that maximum range. [Using the bisection method is advised.] Take the same parameters as in (1), and compute the optimal angle to within 1/10 of a degree in each case. In addition, estimate the corresponding accuracy of the computed maximum range. In view of these results and the numerical technique employed, is your choice of Δt justified?

Plot:

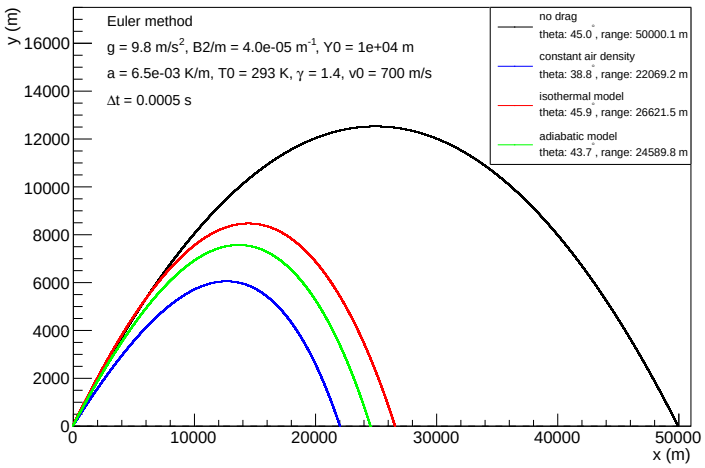

Figure 3: Euler approximation for the cannon shell projectile with different air drag models. The cases shown in this plot have the **maximum ranges**. The shooting angle θ is the angle between the initial velocity and the ground. Here, the input $\Delta t = 0.0005$ s.

Physics explanation:

Figure 3 shows the trajectories with the maximum range of each model. The projection angle (between the initial velocity and the ground) and maximum range of each model are listed in the legend on the top-right corner. We use the bisection method to search for the maximum range. The algorithm can be briefly described as following.

(1) Choose an initial point (θ_0) and step size ($\Delta\theta_0$) as the current θ and $\Delta\theta$.

(2) From the current θ and $\Delta\theta$, we get the left point ($\theta_l = \theta - \Delta\theta$) and the right point ($\theta_r = \theta + \Delta\theta$). We regard the range R as a function of the projection angle θ, $R(\theta)$. We compare $R(\theta)$, $R(\theta_l)$, and $R(\theta_r)$ and get the maximum one of the three.
If $R(\theta)$ is the maximum one, we halve the step size $\Delta\theta \to \Delta\theta/2$ and repeat this step.
If $R(\theta_l)$ (or $R(\theta_r)$) is the maximum one, we choose θ_l (or θ_r) as the new current point and repeat this step.

(3) When the step size is small enough, we end the search and regard the current point θ and $R(\theta)$ as the optimal.

The corresponding code will be attached later.

In the bisection search procedure above, we can estimate the accuracy (or uncertainty) of the maximum range from the difference between $R(\theta)$ and $R(\theta_l)$ (or $R(\theta_r)$) right before existing the loop. For my case, I will choose $\Delta\theta_0 = 32°$, so the step size would be $\Delta\theta = 0.0625°$ finally.

model	θ	range (m)	$R(\theta) - R(\theta - \Delta\theta)$ (m)	$R(\theta) - R(\theta + \Delta\theta)$ (m)
(a) no drag	45.0°	50000.1	0.1187	0.1193
(b) constant air density	38.8°	22069.2	0.0751	0.0012
(c) isothermal model	45.9°	26621.5	0.0599	0.0423
(d) adiabatice model	43.7°	24589.8	0.0286	0.0565

Table 1: The projection angle θ and range of the case with the maximum range. The left and right uncertainties are estimated.

From Table 1, we can see that the **accuracy** of searching for the maximum range is high. The relative uncertainty is about 3×10^{-6}. The Euler approximation can give comparable error in the range. For example, in the model of no drag, the simple analytical calculation gives the range 50000 m, whereas the Euler approximation gives 50000.1 m. The relative error is about 2×10^{-6}, which is comparable to the relative uncertainty from the maximum range bisection search 3×10^{-6}. As a result, the choice of $\Delta t = 0.0005$ s is **justified** to be suitable.

However, this small step size Δt would be too expensive. The more economical way would be to change the method from Euler approximation to the higher order Runge-Kutta approximation (like RK4) (see Fig. 4 and Fig. 5 in the backup plots). The similar accuracy is achieved by a much larger step size $\Delta t = 0.02$ s. The time complexity is just $4 \times 0.0005/0.02 = 0.1$ of the Euler approximation with $\Delta t = 0.0005$ s.

Relevant code:

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

double Projectile::cal_range() {
        int n = _n_stps - 1;
        //cout << "y[n-1] = " << _x[1][n-1] << "; y[n] = " << _x[1][n] << endl;
        _range = ( _x[1][n]*_x[0][n-1] - _x[1][n-1]*_x[0][n] )/( _x[1][n] - _x[1][n-1] );
        return _range;
}

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

double Projectile::search_theta_for_max_range(double v = 700) {
        double dtheta = 32;
        double theta = 45;
        double theta_tmp_left;
        double theta_tmp_right;
        //double v = 700;
        double range = -1;
        double range_tmp_left;
        double range_tmp_right;

        do {
                theta_tmp_left = theta - dtheta;
                _c_start[2] = v*cos(theta_tmp_left*M_PI/180);
                _c_start[3] = v*sin(theta_tmp_left*M_PI/180);
                cal();
                range_tmp_left = cal_range();

                theta_tmp_right = theta + dtheta;
                _c_start[2] = v*cos(theta_tmp_right*M_PI/180);
                _c_start[3] = v*sin(theta_tmp_right*M_PI/180);
                cal();
                range_tmp_right = cal_range();
```

```
34
35                  if(range_tmp_left <= range && range_tmp_right <= range) {
36                          dtheta /= 2.0;
37                          continue;
38                  }
39                  if(range_tmp_left > range) {
40                          range = range_tmp_left;
41                          theta = theta_tmp_left;
42                  }
43                  if(range_tmp_right > range) {
44                          range = range_tmp_right;
45                          theta = theta_tmp_right;
46                  }
47          } while(dtheta>0.05);
48
49          cout << "left uncertainty: " << range-range_tmp_left << endl;
50          cout << "right uncertainty: " << range-range_tmp_right << endl;
51
52          return theta;
53  }
54
55  //------------------------------------------------------------------------//
```

Backup plots:

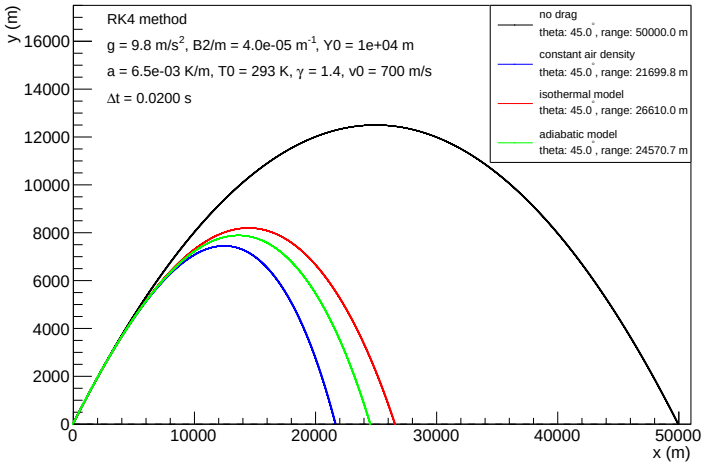

Figure 4: RK4 approximation for the cannon shell projectile with different air drag models. The shooting angle θ is the angle between the initial velocity and the ground. Here, the input $\Delta t = 0.02$ s.

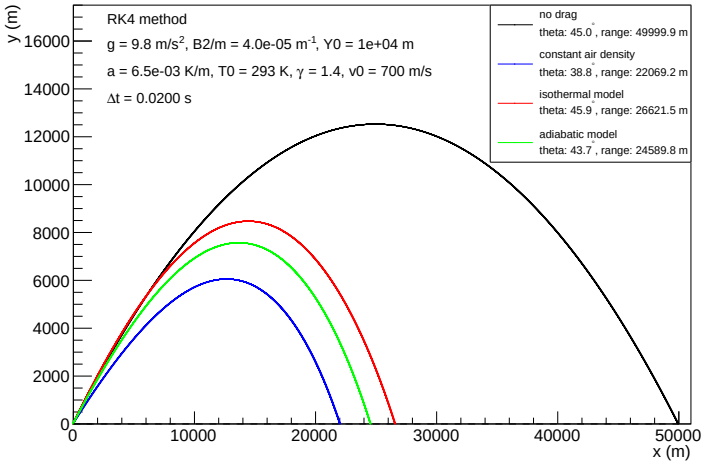

Figure 5: RK4 approximation for the cannon shell projectile with different air drag models. The cases shown in this plot have the **maximum ranges**. The shooting angle θ is the angle between the initial velocity and the ground. Here, the input $\Delta t = 0.02$ s.