物理与计算机的交叉应用探索

普物大作业　计算机图灵实验班　孙浩翔　2023202305

目录

1　引言

本文中，我们希望探讨一些物理学与计算机科学的交叉话题. 物理学是一门严谨的学科. 借助数学与物理学的工具，我们得以使用精确的语言来描述世界. 然而，从三体问题到流体的湍流，混沌系统也为我们对世界的理解罩上了一层乌云.

随着计算机科学的不断发展，人们得以借助计算机来模拟物理现象. 物理学的严谨性与计算机的高效性的有机结合，也为我们带来了新的理论洞见与实际应用. 接下来，笔者将从一名计算机专业本科生的视角，探讨若干物理学与计算机科学的交叉话题.

2　物理现象的计算机模拟

计算机的强大算力为真实物理现象的模拟提供了契机. 接下来，我们将以带阻力的抛体运动模拟为例，探索计算机在物理学中的应用.

2.1　带阻力的抛体运动模拟

我们考虑从原点抛出，初速度为 $\boldsymbol{v}_0$ 的质点在阻力 $\boldsymbol{F}=\boldsymbol{F}(\boldsymbol{v})$ 与重力作用下的轨道.

2.1.1　简单情形：$\boldsymbol{F}=-\alpha\boldsymbol{v}$

对于简单的情形，解析解是可以得到的. 具体地

$$m\frac{\mathrm{d}v_x}{\mathrm{d}t}=F_x=-\alpha v_x \qquad m\frac{\mathrm{d}v_y}{\mathrm{d}t}=-mg+F_y=-mg-\alpha v_y$$

可以解得

$$x=\frac{mv_{0x}}{\alpha}(1-e^{-\alpha t/m}) \qquad y=\left(\frac{m^2g}{\alpha^2}+\frac{mv_{0y}}{\alpha}\right)(1-e^{-\alpha t/m})-\frac{mg}{\alpha}t$$

消去 t 可以得到轨迹方程

$$y=\left(\frac{mg}{\alpha v_{0x}}+\frac{v_{0y}}{v_{0x}}\right)x+\frac{m^2g}{\alpha^2}\ln\left(1-\frac{\alpha x}{mv_{0x}}\right)$$

我们借助 `Python` 进行绘图，取 $m=1$ ，$v_{0x}=3$ ，$v_{0y}=4$ ，$g=9.8$.

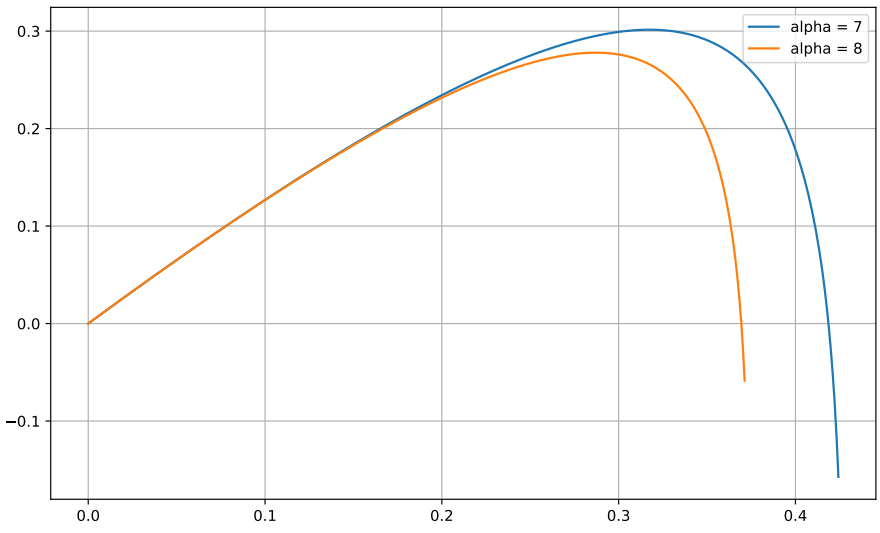


图 1：简单情形的演示

2.1.2 更复杂情形：$\boldsymbol{F}=-\alpha v^{\gamma}\boldsymbol{v}$

接下来我们来研究 $\boldsymbol{F}=-\alpha v^{\gamma}\boldsymbol{v}$ 阻力下的运动曲线.

事实上，即使是 $\boldsymbol{F}=-\alpha v\boldsymbol{v}$ 的情形，我们也只能通过计算机模拟来得到近似解，而无法与简单情形一样得到不含积分的解析解. 接下来我们将借助计算机模拟的方式，画出轨迹的近似图像.

考虑运动微分方程

$$m\frac{\mathrm{d}v_x}{\mathrm{d}t}=F_x \qquad m\frac{\mathrm{d}v_y}{\mathrm{d}t}=-mg+F_y$$

对于足够小的时间 Δt ，可以认为

$$v_{x,n+1}=v_{x,n}-\frac{\alpha v_{x,n}\Delta t}{m}(v_{x,n}^2+v_{y,n}^2)^{\gamma/2}$$

$$v_{y,n+1}=v_{y,n}-\frac{\alpha v_{y,n}\Delta t}{m}(v_{x,n}^2+v_{y,n}^2)^{\gamma/2}-g\Delta t$$

同时我们有 $\Delta x=v_x\Delta t$ ， $\Delta y=v_y\Delta t$. 利用这一点，我们取足够小的时间 Δt ，按上式迭代 N 次，就可以模拟 $N\Delta t$ 时间内的运动.

借助 `Python` ，取 $m=1$ ， $g=9.8$ ，我们可以绘制如下的若干条轨迹：

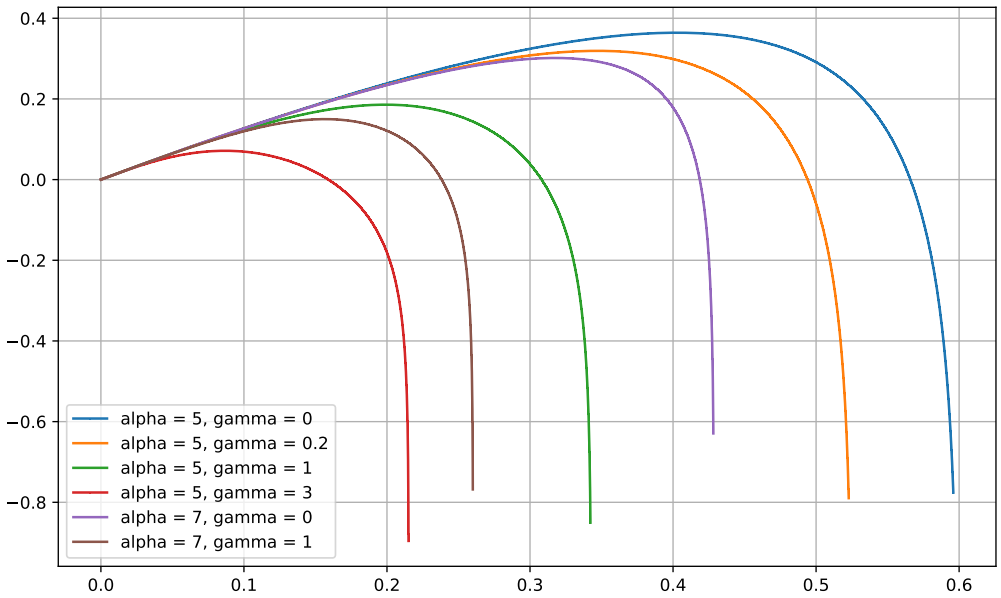


图 2：更复杂情形的演示

2.2 进一步分析

我们来考虑如何使用更精确的方法来解决这个问题. 设运动方向与水平面的夹角为 θ ，运动轨迹的曲率半径为 ρ .

在轨迹切向与法向分别有

$$m\frac{\mathrm{d}v}{\mathrm{d}t} = -\alpha v^{\gamma+1} - mg\sin\theta \qquad m\frac{v^2}{\rho} = mg\cos\theta$$

同时我们有

$$\frac{v\mathrm{d}t}{\rho} = \frac{\mathrm{d}s}{\rho} = \mathrm{d}\theta \qquad \frac{\mathrm{d}v}{\mathrm{d}t} = \frac{\mathrm{d}v}{\mathrm{d}\theta}\frac{\mathrm{d}\theta}{\mathrm{d}t}$$

于是

$$\frac{\mathrm{d}v}{\mathrm{d}\theta} + \frac{\alpha v^{\gamma+2}}{mg\cos\theta} + v\tan\theta = 0$$

借助 `Wolfram Mathematica` ，我们可以求得上述微分方程的解

$$v(\theta) = \left((v_0\cos\theta)^{-1-\gamma}\cos^{1+\gamma}\theta_0 + \frac{\alpha}{mg\mathrm{sgn}(\sin\theta)}(f(\theta)+g(\theta))\right)^{-\frac{1}{1+\gamma}}$$

$$f(\theta) = -\mathrm{F}\left(\frac{1}{2}, \frac{1+\gamma}{2}, \frac{3+\gamma}{2}, \cos^2\theta\right)$$

$$g(\theta) = \mathrm{F}\left(\frac{1}{2}, \frac{1+\gamma}{2}, \frac{3+\gamma}{2}, \cos^2\theta_0\right)\mathrm{sgn}(\sin\theta\sin\theta_0)(\cos\theta_0\sec\theta)^{\gamma+1}$$

其中 $\mathrm{sgn}x$ 为符号函数，$\mathrm{F}(a,b,c,z)$ 为超几何级数

$$\mathrm{F}(a,b;c;z) = \sum_{n=0}^{\infty}\frac{(a)_n(b)_n}{(c)_n}\frac{z^n}{n!} = 1 + \frac{ab}{c}\frac{z}{1!} + \frac{a(a+1)b(b+1)}{c(c+1)}\frac{z^2}{2!} + \cdots$$

借助笔者已知的数学工具与软件，无法再进行进一步的分析. 这提示我们计算机模拟的重要性. 从图像上来看，各轨迹似乎具有渐近线 $x = x_0$. 通过计算机模拟，我们可以得到 x_0 的近似取值. 笔者所尝试的几组值见下表，取 $m = 1$ ， $g = 9.8$ ， $\Delta t = 10^{-5}$.

序号	α	γ	x_0
1	3	0	1.0000
2	3	1	0.5162
3	3	2	0.3461
4	5	1	0.3428
5	5	2	0.2510

3 基于物理定律的计算机算法

计算机的强大算力使得我们可以更直观地理解物理现象. 与此同时，物理定律也为计算机算法的设计提供了思路. 接下来，我们将以基于相互作用的图排布与模拟退火算法为例，探索物理在计算机科学中的应用.

3.1 基于相互作用的图排布

考虑图 $G = (V, E)$. 在平面上，G 可以有无数多种排布. 例如，下面的图中展示了图 G 的几种不同的排布. （注：同构是指点之间的连边关系一致. 例如，下图中的三张图都具有边 $A - D$, $A - C$, $C - D$, $A - E$, $C - B$. 形象地说，通过拖拽下图中某一张图的点，就可以得到其他两张图）

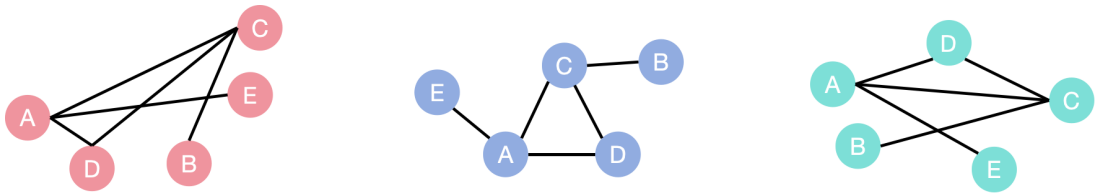


图 3：几种不同排布的同构图

如图所示，中间的一张图具有“更清晰”的排版. 具体来说，边与边之间的夹角更大，交叉更少. 这提示我们借助相互作用对图上的点与边进行合理的排布，使得最终展示出的示意图最清晰. 下图是更复杂的一个例子，左右两边的图也是同构的.

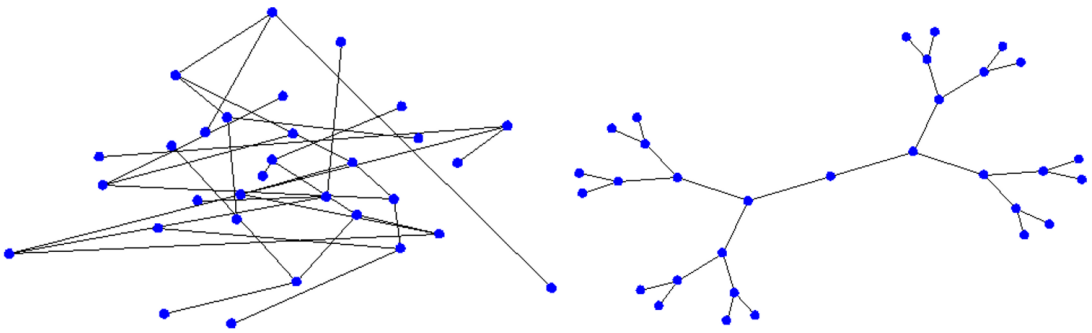


图 4：两种不同排布的同构图

在这里，我们将采用如下的策略：首先，每一条边 $e : u \to v$ 将产生拉力，使得 u, v 更加靠近. 这里的拉力我们选用胡克定律，定义为

$$F_1 = k_1 d$$

其中 $d = d(u, v)$ 表示 u, v 在平面上的距离. 其次，每两个点之间将具有斥力，使得 u, v 距离更远. 这里的斥力我们选用库仑定律，定义为

$$F_2 = \frac{k_2}{d^2}$$

于是我们迭代 N 次，每次计算两两结点之间力的作用，更新位置即可. 采用这两个策略，我们可以使整张图的分布较为平均. 下图是一个示例，最左侧是未经处理的排布，最右侧是经过足够多次迭代后得到的排布.

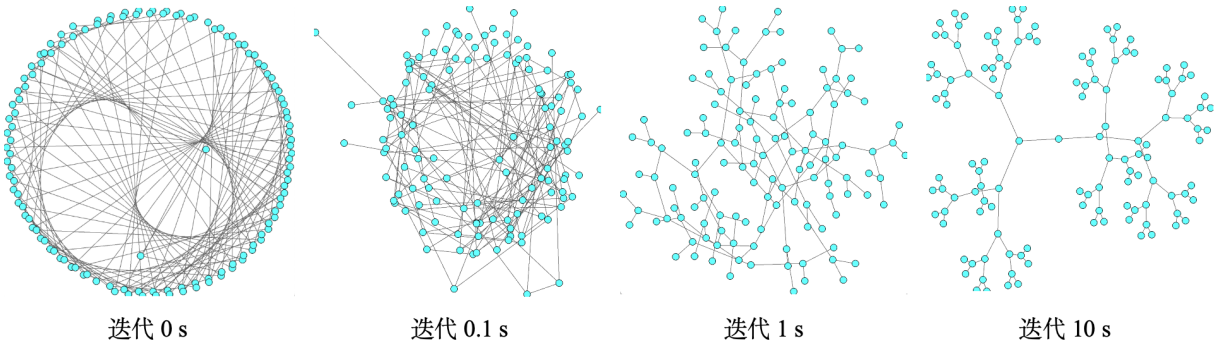
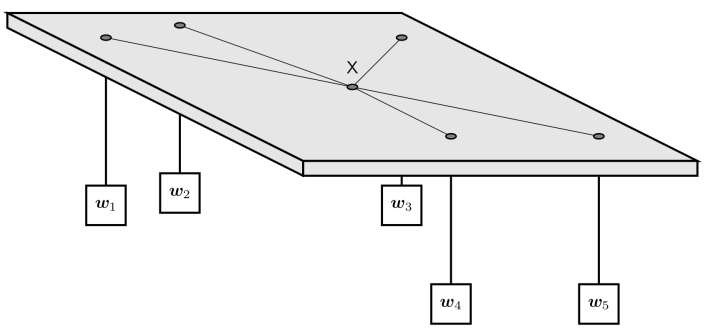


图 5：基于相互作用的排布演示

3.2 模拟退火算法

“退火” 是将材料加热后再经特定速率冷却的技术. 材料中的原子原本停留在使内能有局部最小值的位置，我们通过加热使原子离开原来的位置，而随机在其他位置中移动. 退火冷却时速度较慢，使得原子有较大可能找到内能比原先更低的位置.

模拟退火算法就是以该物理过程为基础发展而来的随机算法. 简单的说，我们定义全局温度 T . 每一时刻，该状态以概率 $p(T)$ 移动到能量更低的状态. 与此同时，温度逐渐下降，使得系统越来越倾向于接受能量较低的状态，即 $p(T)$ 变大. 利用这一种算法，我们将更不容易陷入局部最优解. 接下来，我们将以下面的这个问题为例，探索模拟退火算法的应用.



如图所示，n 个重为 m_k 的重物分别系在足够长的轻质绳上. 每条绳子自上而下穿过桌面上的洞，然后系在一起，公共绳结位于 x 处. 假设桌子足够高，且忽略所有的摩擦，问绳结 x 最终平衡于何处.

首先我们来分析这一问题. 不妨设物体到 x 的绳长均为 L ，L 足够大. 我们使用 d_k 表示 x 到 m_k 所对应洞口的距离. 取桌面为零势能面，我们考虑静止时系统的重力势能

$$E_p = \sum_{k=1}^{n} -m_k g(L - d_k) = g\sum_{k=1}^{n} m_k d_k - gL\sum_{k=1}^{n} m_k = g\sum_{k=1}^{n} m_k d_k - C$$

注意到 E_p 应具有最小值. 换言之，我们可以定义这个状态具有的能量

$$E = \sum_{k=1}^{n} d_k m_k$$

最终的 x 的位置应满足 E 最小. 我们直接使用模拟退火算法求解即可. 代码如下：

```
import numpy as np
N = 1009
D = 0.97
EPS = 1e-14
n = int(input())
x = np.zeros(n)
y = np.zeros(n)
w = np.zeros(n)
for i in range(n):
    x[i], y[i], w[i] = map(float, input().split())
def calc(x0, y0):
    res = 0
    for i in range(n):
        res += np.sqrt((x[i] - x0) ** 2 + (y[i] - y0) ** 2) * w[i]
    return res
bx, by = np.mean(x), np.mean(y)
best = calc(bx, by)
for iteration in range(100):
    ans = best
    x0, y0 = bx, by
    T = 100000
    while T > EPS:
        x1 = x0 + T * (np.random.rand() * 2 - 1)
        y1 = y0 + T * (np.random.rand() * 2 - 1)
        res = calc(x1, y1)
        if res < best:
            best = res
            bx, by = x1, y1
        if res < ans or np.exp((ans - res) / T) > np.random.rand():
            ans = res
            x0, y0 = x1, y1
        T *= D
print(f"{bx:.3f} {by:.3f}")
```

3.2.1 使用模拟退火逼近最速降线

借助计算机模拟方法，我们可以试图找到最速降线. 下面，我们将以 $A(0,1)$ 下落到 $B(1,0)$ 为例，演示模拟退火算法在逼近最速降线中的应用.

我们将通过数值模拟的方法来逼近最速降线. 具体地说，我们将 $A(0,1)$ 到 $B(1,0)$ 按横坐标分为 N 份，每一份为长为 $\Delta x = 1/N$ ，高为 $\Delta y_k = y_k - y_{k+1}$ 的轨道. 其中 $y_0 = 1$ ， $y_N = 0$. 通过如下图所示的方法，对 t_k 求和就可以计算出在轨道上总的耗时.

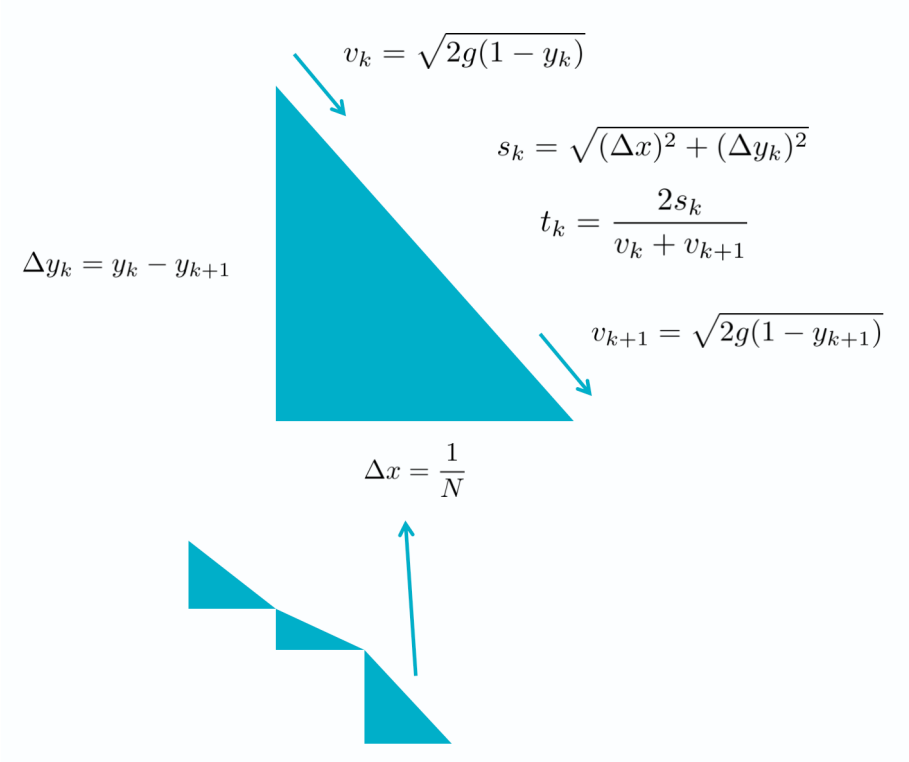


图 6：计算原理

现在，我们需要找到一组 $\{y_k\}$ ，使得 $t = \sum t_k$ 最小. 这一问题可以使用模拟退火算法进行优化. 通过运行附录中的代码，可以得到如下的结果，其中我们选取从 A 到 B 的直线作为迭代的初始位置，蓝色的线为算法迭代一定次数后的优化路径，绿色的线为理论上的最速降线（摆线）.

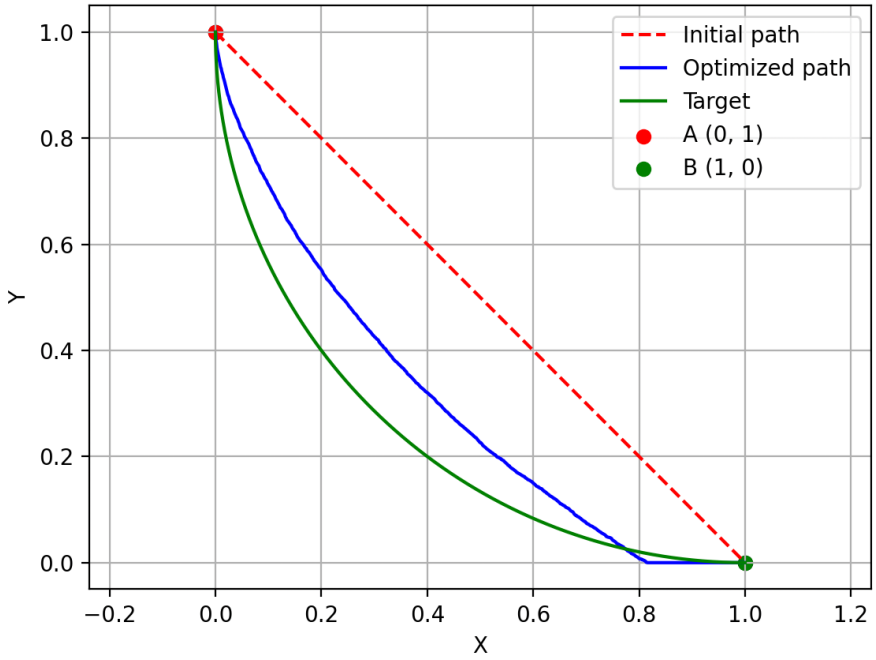


图 7：结果演示

A 代码展示

图 1 的绘制代码

```
import numpy as np
import matplotlib.pyplot as plt

m = 1
g = 9.8
v0x = 3
v0y = 4

def plot(alpha):
    x_max = m * v0x / alpha
    x = np.linspace(0, x_max * 0.99, 400)
    a = m * g / (alpha * v0x) + v0y / v0x
    b = np.log(1 - alpha * x / (m * v0x))
    y = a * x + (m ** 2 * g) / (alpha ** 2) * b
    plt.plot(x, y, label=f'alpha = {alpha}')

plt.figure(figsize=(10, 6))
plot(7)
plot(8)
plt.legend()
plt.grid(True)
plt.show()
```

图 2 的绘制代码

```python
import matplotlib.pyplot as plt

m = 1
g = 9.8

def plot(max_iter, alpha, gamma):
    vx = [3] + [0] * max_iter
    vy = [4] + [0] * max_iter
    x = [0] + [0] * max_iter
    y = [0] + [0] * max_iter
    dt = 0.0001
    for i in range(max_iter):
        a = alpha * dt * ((vx[i] ** 2 + vy[i] ** 2) ** (gamma / 2)) / m
        vx[i + 1] = vx[i] - a * vx[i]
        vy[i + 1] = vy[i] - g * dt - a * vy[i]
        x[i + 1] = x[i] + vx[i] * dt
        y[i + 1] = y[i] + vy[i] * dt
    plt.plot(x, y, marker=',', label=f'alpha = {alpha}, gamma = {gamma}')
    return x, y

plt.figure(figsize=(10, 6))
plot(10000, 5, 0)
plot(10000, 5, 0.2)
plot(10000, 5, 1)
plot(10000, 5, 3)
plot(10000, 7, 0)
plot(10000, 7, 1)
plt.legend()
plt.grid(True)
plt.show()
```

图 5 的绘制代码

```cpp
/*
 * GraphViz: An application that can draw graphs
 * based on input files.
 *
 * @author: CoderBak
 */
#include <iostream>
#include <sstream>
#include <fstream>
#include <cmath>
#include <chrono>
#include "SimpleGraph.h"

using namespace std;

const double pi = 3.14159265358979323;

// Prompt the user to enter input file.
```

```
ifstream fileInput() {
    while (true) {
        string file_name;
        cout << "Please enter the name of a file that contains graph data: ";
        if (!getline(cin, file_name)) {
            throw domain_error("Error when reading file name.");
        }
        ifstream ifs(file_name);
        if (!ifs.fail()) {
            return ifs;
        }
        cout << "Can't read file.\n";
    }
}

// Read an positive integer.
int getInteger() {
    while (true) {
        cout << "Please enter the number of microseconds to run the algorithm: ";
        string line;
        int result;
        char trash;
        if (!getline(cin, line)) {
            throw domain_error("Error when using getline.");
        }
        istringstream iss(line);
        if ((iss >> result) && !(iss >> trash)) {
            if (result > 0) {
                return result;
            }
        }
        cout << "Invalid number.\n";
    }
}

// Get whether the user wants to restart.
string getString() {
    while (true) {
        cout << "Do you want to try another file? Enter yes or no: ";
        string line;
        string result;
        char trash;
        if (!getline(cin, line)) {
            throw domain_error("Error when using getline.");
        }
        istringstream iss(line);
        if ((iss >> result) && !(iss >> trash) && (result == "yes" || result == "no")) {
            return result;
        }
        cout << "Invalid input.\n";
```

```
    }
}

bool checkLoop() {
    string userInput = getString();
    return (userInput == "yes") ? 1 : 0;
}

void initialize(SimpleGraph& G, const size_t node_cnt, ifstream& file) {
    G.nodes.clear();
    G.edges.clear();
    for (size_t k = 0; k < node_cnt; ++ k) {
        double x = cos(2 * pi * k / node_cnt);
        double y = sin(2 * pi * k / node_cnt);
        G.nodes.push_back({x, y});
    }
    size_t u, v;
    while (file >> u) {
        file >> v;
        G.edges.push_back({u, v});
    }
}

void edgeAdjust(SimpleGraph& G) {
    const double kattract = 0.005;

    for (auto edge : G.edges) {
        auto [u, v] = edge;
        auto [x0, y0] = G.nodes[u];
        auto [x1, y1] = G.nodes[v];
        double Fattract = kattract * sqrt(pow(y1 - y0, 2) + pow(x1 - x0, 2));
        double theta = atan2(y1 - y0, x1 - x0);
        G.nodes[u].x += Fattract * cos(theta);
        G.nodes[u].y += Fattract * sin(theta);
        G.nodes[v].x -= Fattract * cos(theta);
        G.nodes[v].y -= Fattract * sin(theta);
    }
}

void nodeAdjust(SimpleGraph& G, const size_t node_cnt) {
    const double krepel = 0.005;

    for (size_t i = 0; i < node_cnt; ++ i) {
        for (size_t j = i + 1; j < node_cnt; ++ j) {
            auto [x0, y0] = G.nodes[i];
            auto [x1, y1] = G.nodes[j];
            double Frepel = krepel / (pow(y1 - y0, 2) + pow(x1 - x0, 2));
            double theta = atan2(y1 - y0, x1 - x0);
            G.nodes[i].x -= Frepel * cos(theta);
            G.nodes[i].y -= Frepel * sin(theta);
```

```
            G.nodes[j].x += Frepel * cos(theta);
            G.nodes[j].y += Frepel * sin(theta);
        }
    }
}

int main() {
    while (true) {
        ifstream file = fileInput();
        int timeLimit = getInteger();
        auto start = chrono::high_resolution_clock::now();
        SimpleGraph G;
        size_t node_cnt;
        file >> node_cnt;
        initialize(G, node_cnt, file);
        DrawGraph(G);
        while (true) {
            auto end = chrono::high_resolution_clock::now();
            auto elapsed = chrono::duration_cast<std::chrono::milliseconds>(end - start);
            if (elapsed.count() * 1000 > timeLimit) {
                break;
            }
            edgeAdjust(G);
            nodeAdjust(G, node_cnt);
            DrawGraph(G);
        }
        DrawGraph(G);
        if (!checkLoop()) {
            break;
        }
    }
    return 0;
}
```

图 7 的绘制代码

```
import numpy as np
import matplotlib.pyplot as plt

g = 9.81

def calculate_time(path):
    time = 0
    for i in range(len(path) - 1):
        x1, y1 = path[i]
        x2, y2 = path[i + 1]
        dx = x2 - x1
        dy = y2 - y1
        ds = np.sqrt(dx ** 2 + dy ** 2)
        v1 = np.sqrt(2 * g * (1 - y1))
        v2 = np.sqrt(2 * g * (1 - y2))
```

```
        time += 2 * ds / (v1 + v2)
    return time


def perturb_path(path, scale=2):
    new_path = path.copy()
    idx = np.random.randint(1, len(path) - 1)
    perturbation = scale * np.random.uniform(-1, 1)
    new_path[idx][1] += perturbation
    new_path[idx][1] = max(new_path[idx][1], 0)
    new_path[idx][1] = min(new_path[idx][1], 1)
    new_path[0] = np.array([0, 1])
    new_path[-1] = np.array([1, 0])
    ys = [_[1] for _ in new_path]
    ys = sorted(ys, key=lambda x: -x)
    new_path = [[k / 1000, ys[k]] for k in range(1000)]
    return np.array(new_path)

def simulated_annealing(initial_path, initial_temp, cooling_rate, max_iter):
    current_path = initial_path
    current_temp = initial_temp
    best_path = current_path
    best_time = calculate_time(current_path)
    for i in range(max_iter):
        new_path = perturb_path(current_path)
        new_time = calculate_time(new_path)
        if new_time < best_time or np.exp((best_time - new_time) / current_temp) > np.random.rand():
            current_path = new_path
            best_time = new_time
            best_path = current_path
        current_temp *= cooling_rate
        if current_temp < 1e-10:
            break
    return best_path, best_time

num_points = 1000
initial_path = np.linspace([0, 1], [1, 0], num_points)

# SA parameters
initial_temp = 100000
cooling_rate = 0.999
max_iter = 10000000

best_path, best_time = simulated_annealing(initial_path, initial_temp, cooling_rate, max_iter)

plt.plot(initial_path[:, 0], initial_path[:, 1], 'r--', label='Initial path')
plt.plot(best_path[:, 0], best_path[:, 1], 'b-', label='Optimized path')
plt.scatter([1, 0], [0, 1], c='black')

start = [0, 1]
```

```
end = [1, 0]


theta = np.linspace(0, np.pi / 2, 1000)
x = start[0] + (end[0] - start[0]) * (1 - np.cos(theta))
y = start[1] + (end[1] - start[1]) * np.sin(theta)
plt.plot(x, y, label='Target', color='g')
plt.scatter(start[0], start[1], color='r', label='A (0, 1)')
plt.scatter(end[0], end[1], color='g', label='B (1, 0)')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid(True)
plt.axis('equal')
plt.show()
```