Training in Algorithmic Programming

Engineering Mechanics – Aeronautics and Astronautics

Fluid Dynamics

Particle Motion Simulation
Trajectory Calculation
Velocity Distribution

Zhejiang University · Hangzhou · China

计算程序设计训练大作业

张柏铭　3230100298

2025 年 6 月 28 日

目录

Part I

两相横流问题

1 问题重述

1.1 流体控制方程

流体的速度分布为：

$$u_{\text{fluid}} = U_{\max}\left(1 - y^2\right) \tag{1}$$

$$v_{\text{fluid}} = 0 \tag{2}$$

其中，$U_{\max}$ 表示中心最大槽流速，设为 5。

1.2 颗粒运动控制方程

颗粒运动的控制方程为：

$$\begin{cases} m_s \dfrac{\mathrm{d}u_s}{\mathrm{d}t} = F_{dx} \\ m_s \dfrac{\mathrm{d}v_s}{\mathrm{d}t} = F_{dy} \\ \overrightarrow{F_d} = \dfrac{1}{2} C_d \rho_{\text{fluid}} \left| \overrightarrow{V_{\text{fluid}}} - \overrightarrow{V_s} \right| (\overrightarrow{V_{\text{fluid}}} - \overrightarrow{V_s}) A_s + m\overrightarrow{g} \end{cases} \tag{3}$$

1.3 参数

符号	单位	描述	公式/取值
r	m	颗粒半径	0.01
ρ_s	$\mathrm{kg\,m^{-3}}$	颗粒密度	2.8
ρ_{fluid}	$\mathrm{kg\,m^{-3}}$	流体密度	1.0
C_d	–	阻力系数	0.50
A_s	$\mathrm{m^2}$	横截面积	πr^2
m_s	kg	颗粒质量	$\frac{4}{3}\pi r^3 \rho_s$
$\vec{V}_{\text{fluid}}$	$\mathrm{m\,s^{-1}}$	流体速度	$[U_{\max}(1-y^2), 0]^T$
$\vec{V}_s$	$\mathrm{m\,s^{-1}}$	颗粒速度	需要求解
$U_{\max}$	$\mathrm{m\,s^{-1}}$	最大流速	5

表 1: 参数与符号描述表

1.4 初始条件

1. 颗粒从距入口 2.5 的位置以速度 1.0 垂直于主流进入；

2. 颗粒从水槽左侧入口（0, 0.40），（0, 0），（0, -0.40）以速度 1.6 平行于主流进入槽道。

2 核心代码展示

```
% Not ploting every point, so we can determine the velocity by
    observing the density of the points.
sample_rate = 4;
% Fluid velocity profile
fluid_velocity = @(y) [U_max * (1 - y^2); 0];
% Drag force calculation
drag_force = @(v_particle, v_fluid) ...
    0.5 * C_d * density_fluid * norm(v_fluid - v_particle) * ...
    (v_fluid - v_particle) * area;

% ===== Example Simulation =====
% IC1 Simulation
positions_1 = zeros(2, num_steps + 1);
positions_1(:, 1) = initial_position_1;
velocity = initial_velocity_1;
for t = 1:num_steps
    v_fluid = fluid_velocity(positions_1(2, t));
    drag = drag_force(velocity, v_fluid);
    gravity = [0; -mass * g];
    acceleration = (drag + gravity) / mass;
    velocity = velocity + acceleration * dt;
    positions_1(:, t + 1) = positions_1(:, t) + velocity * dt;
end
```

详细代码请见附录A：两相横流问题的MATLAB完整代码实现[1]。

3 结果分析

3.1 轨迹展示

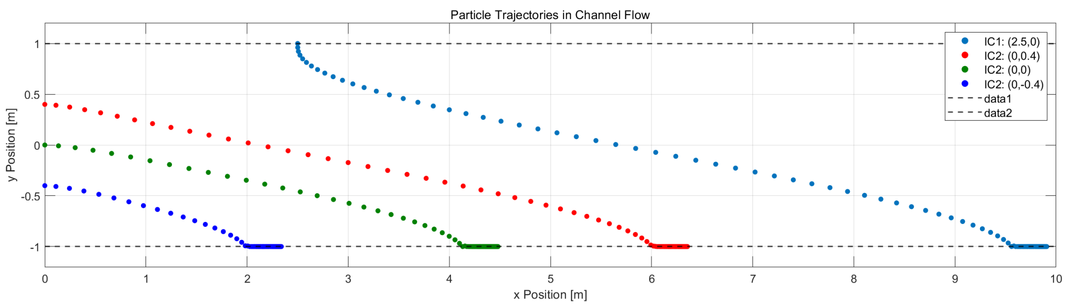

图 1: 不同初始条件颗粒的轨迹展示

[1] 由于Latex环境下Matlab代码注释不支持中文（会乱码），所以以下均采用英文注释。

3.2 初始条件1

颗粒从距入口2.5的位置以速度-1.0垂直于主流进入。

(a) 前半段：因受重力，且越往中心 y=0 水流速度越大，带动颗粒速度逐渐增大。

(b) 后半段：到中间左右颗粒又因为阻力较大而减速，直至碰壁后损失竖直方向速度。

(c) 终态：水平方向因为惯性还会继续向右一段路，但是由于一直受到阻尼力而不断减速，最终趋于静止。

3.3 初始条件2

颗粒从水槽左侧入口的三个不同位置（0, 0.40）、（0, 0）、（0, -0.40）以速度1.6平行于主流进入槽道。

(a) 前半段：因受重力，越往下颗粒速度逐渐增大。

(b) 后半段：颗粒因为阻力较大而减速，直至碰壁后损失竖直方向速度。

(c) 终态：水平方向因为惯性还会继续向右一段路，但是由于一直受到阻尼力而不断减速，最终趋于静止。

3.4 总结

综上所述，通过数值模拟可以清晰地观察到颗粒在两相横流中的运动轨迹，并得出结论：初始越上方入射的颗粒，一般来说越晚沉积，沉积位置越靠右。

Part II

颗粒轨迹

1 问题重述

1.1 情景设置

需要模拟一个颗粒在二维平面（x-y）中的运动轨迹，其速度由以下方程给出：

$$\begin{cases} v_x(t) = \dfrac{dx}{dt} = \cos(6t), \\ v_y(t) = \dfrac{dy}{dt} = \sin(6t), \end{cases} \tag{4}$$

其中t为时间，颗粒初始位置为原点$(x, y) = (0, 0)$。

1.2 函数实现

编写函数`ComputeTrajectory`，要求：

- 输入参数：
  - `dt`：时间步长（单位：秒）
  - `timesteps`：总模拟步数
- 输出结果：
  - 返回$2 \times$ `timesteps`矩阵`state`
  - 第一行为x坐标序列，第二行为y坐标序列

1.3 测试任务

编写测试脚本完成：

- 设置参数：`dt = 0.005`，`timesteps = 2000`
- 调用函数计算轨迹
- 绘制y关于x的轨迹图，并标注起点$(0, 0)$

2 核心代码

这就是一个很简单的离散积分问题，这里分别采用Euler法和四阶Runge-Kutta法。

2.1 Euler法

```
function state = ComputeTrajectory(dt, timesteps)
    % Initialize state matrix (2 rows x timesteps columns)
    state = zeros(2, timesteps);

    for k = 1:timesteps
        % Calculate velocity at current timestep
        vx = cos(6 * (k-1) * dt);  % t = (k-1)*dt
        vy = sin(6 * (k-1) * dt);

        % Update position using Euler integration
        x = x + vx * dt;
        y = y + vy * dt;

        % Store current state
        state(:, k) = [x; y];
    end
end
```

2.2 四阶Runge-Kutta法

```
function state = ComputeTrajectory(dt, timesteps)
    % Initialize state matrix (2 rows x timesteps columns)
    state = zeros(2, timesteps);
    for k = 1:timesteps
        t = (k-1) * dt;  % Current simulation time
        % -- 4th-order Runge-Kutta method --
        % k1: velocity at beginning of time step
        k1_x = cos(6 * t);                  k1_y = sin(6 * t);
        % k2: velocity at midpoint using k1
        x_temp = x + (dt/2) * k1_x;         y_temp = y + (dt/2) * k1_y;
        k2_x = cos(6 * (t + dt/2));         k2_y = sin(6 * (t + dt/2));
        % k3: improved midpoint velocity using k2
        x_temp = x + (dt/2) * k2_x;         y_temp = y + (dt/2) * k2_y;
        k3_x = cos(6 * (t + dt/2));         k3_y = sin(6 * (t + dt/2));
        % k4: velocity at end of time step using k3
        x_temp = x + dt * k3_x;             y_temp = y + dt * k3_y;
        k4_x = cos(6 * (t + dt));           k4_y = sin(6 * (t + dt));
        % Update position using weighted average of k1-k4
        x = x + (dt/6) * (k1_x + 2*k2_x + 2*k3_x + k4_x);
        y = y + (dt/6) * (k1_y + 2*k2_y + 2*k3_y + k4_y);
        % Store current state
        state(:, k) = [x; y];
    end
end
```

详细代码请见附录B：颗粒轨迹的MATLAB完整代码实现。

3 结果分析

3.1 轨迹展示

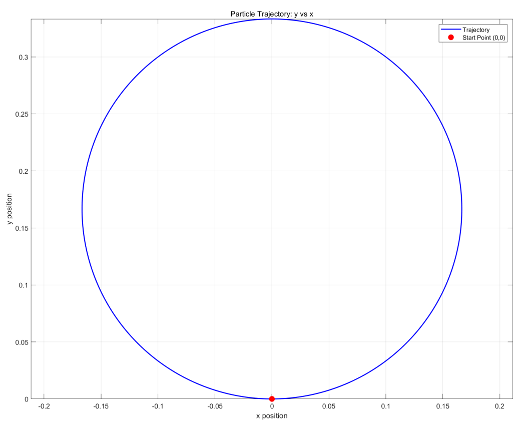

图 2: 颗粒轨迹示意图

3.2 图片解释

精确积分可得：

$$\begin{cases} x(t) = \dfrac{1}{6}\sin(6t), \\ y(t) = \dfrac{1}{6}(1-\cos(6t)). \end{cases} \tag{5}$$

消去参数 t，可得轨迹一个圆心为$(0, \frac{1}{6})$，半径为$\frac{1}{6}$的圆：

$$x^2 + \left(y - \frac{1}{6}\right)^2 = \left(\frac{1}{6}\right)^2 \tag{6}$$

1. 颗粒以角速度 $6\,\mathrm{rad/s}$ 做匀速圆周运动。
2. 速度大小恒为 $\sqrt{\cos^2(6t) + \sin^2(6t)} = 1$（单位速度）。

由此可见，在精度较高的情况下（t=0.005）时，数值模拟和实际积分结果相当接近，同时Euler法和四阶Runge-Kutta法图形基本一致，符合预期。

Part III

水箱内液体流出

1 问题重述

1.1 情景设计

倒圆锥形水箱底部有一半径为 $r_h = 0.025\,\mathrm{m}$ 的圆孔。液体流出速度 v 由托里拆利定理给出：

$$v = \sqrt{2gy}$$

其中 y 为液体高度，$g = 9.81\,\mathrm{m/s^2}$。液体高度随时间变化的微分方程为：

$$\frac{dy}{dt} = -\frac{vr_h^2}{(2-0.5y)^2}$$

初始条件 $y(0) = 2.5\,\mathrm{m}$。

1.2 编程要求

1. 定义匿名函数 `dydt` 表示 $\frac{dy}{dt}$；
2. 在 $t \in [0, 3000]\,\mathrm{s}$ 内用 `ode45` 求解；
3. 绘制 $y(t)$ 直到首次 $y < 0.1\,\mathrm{m}$。

2 关键代码

```
%% Define the Differential Equation (Anonymous Function)
% The negative sign indicates decreasing height over time
dydt = @(t,y) - (sqrt(2*g*y) * r_h^2) / (2 - 0.5*y)^2;

%% Time Array Setup
tspan = 0:dt:t_end; % From 0 to 3000s with step size 0.1s

%% Solve the Differential Equation Using ode45
% Set relative tolerance for improved accuracy
options = odeset('RelTol',1e-6);
[t, y] = ode45(dydt, tspan, y0, options);

%% Find When Liquid Height First Falls Below 0.1m
empty_index = find(y < 0.1, 1);
if isempty(empty_index)
    empty_index = length(t);
    warning('Liquid did not drain below 0.1m within simulation time')
        ;
else
    fprintf('Time when liquid height first falls below 0.1m: %.1f
        seconds\n', t(empty_index));
end
```

详细代码请见附录C：水箱内液体流出的MATLAB完整代码实现。

3 结果分析

3.1 图片展示

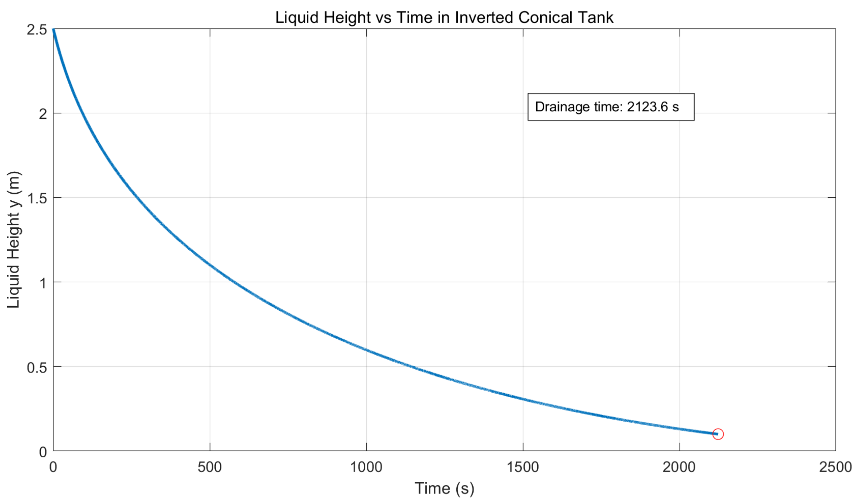

图 3: 倒圆锥水箱液体高度随时间变化

由图可见，t在2123.6秒时液体高度首次低于0.1 m。

3.2 初步分析验证

能量守恒

排出速度 $v = \sqrt{2gy}$ 符合机械能守恒（势能 $\rightarrow$ 动能）。

几何关系

水箱半径随高度线性变化符合倒圆锥形状：$R(y) = 2 - 0.5y$

曲线趋势

- 初始阶段快速下降（小截面积 + 高排出速度）
- 后期阶段下降减缓（大截面积 + 低排出速度）

初步定性判断结果合理。

3.3 定量分析

化简原式：

$$\sqrt{2g}r_h^2 dt = -\frac{(2-0.5y)^2}{\sqrt{y}}dy = -(4y^{-\frac{1}{2}} - 2y^{\frac{1}{2}} + \frac{1}{4}y^{\frac{3}{2}})dy$$

积分有精确解：

$$\sqrt{2g}r_h^2 t = -8y^{\frac{1}{2}} + \frac{4}{3}y^{\frac{3}{2}} - \frac{1}{10}y^{\frac{5}{2}} + C$$

由于没有五次的求根公式，较难求解$y = y(t)$，所以这里采取$t = t(y)$来求解。

$$t(y) = \frac{1}{\sqrt{2g}r_h^2}[8(y_0^{\frac{1}{2}} - y^{\frac{1}{2}}) - \frac{4}{3}(y_0^{\frac{3}{2}} - y^{\frac{3}{2}}) + \frac{1}{10}(y_0^{\frac{5}{2}} - y^{\frac{5}{2}})]$$

其中$y_0 = 2.5m$为初始高度，$r_h = 0.025m$为小孔半径，$g = 9.81m/s^2$为重力加速度。

画得理论轨迹如下：（代码见附录C_1：理论y-t关系的MATLAB完整代码实现。）

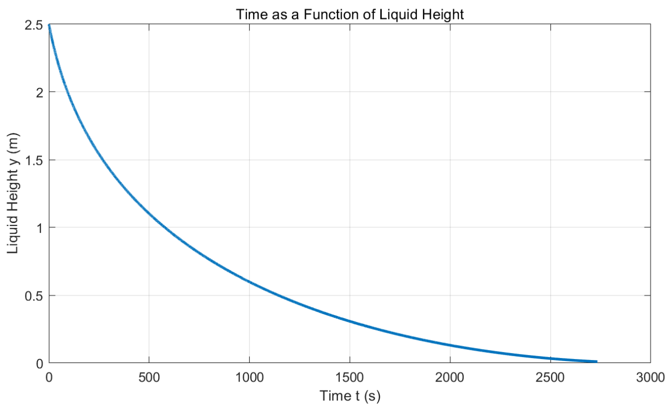

图 4: 理论y-t关系

3.4 误差分析

那既然两个都做了，肯定要把它画到一起：

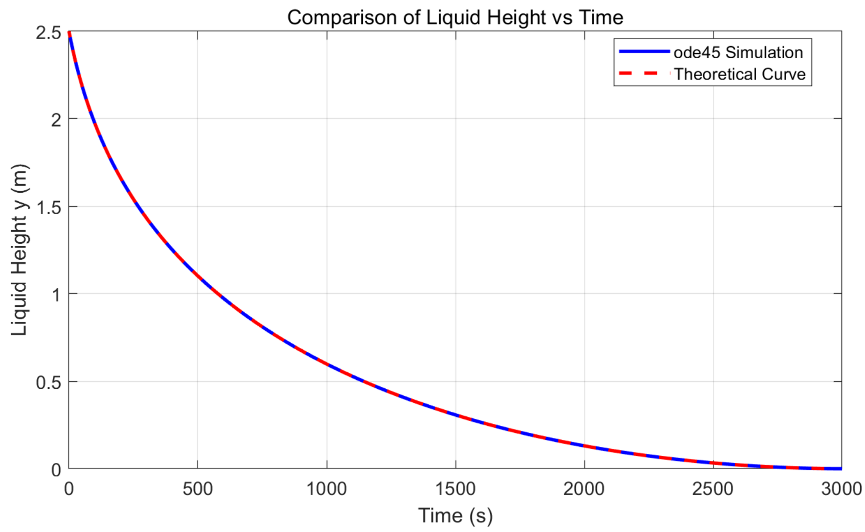

图 5: 理论和迭代方法对比

然后计算它们的误差：

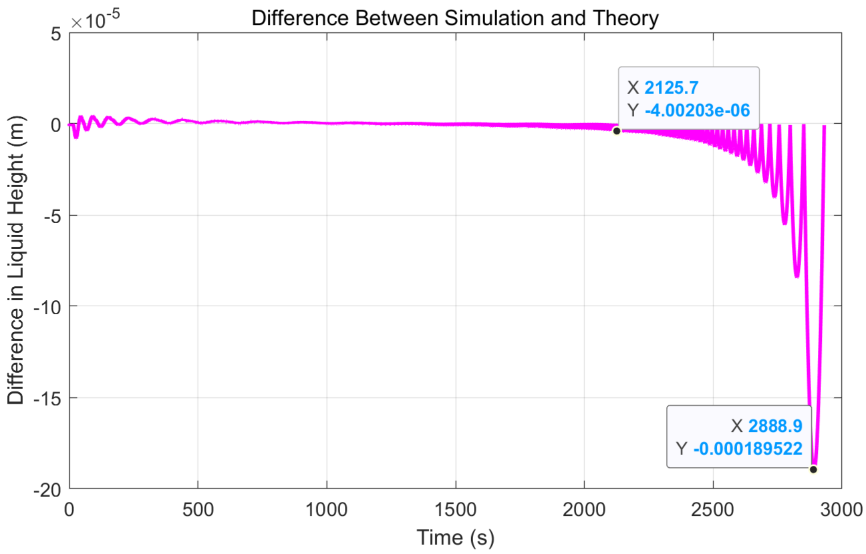

图 6: 误差分析

其中我标出了t在题目要求绘制极限的误差约为4.00203^{-6}，相对误差为1.600812^{-6}。

求解域内最大误差为0.000189522，最大相对误差为0.0000758088大约万分之一级别，可见该方法误差极小，使用合理。

详细代码请见附录C_2：水箱内液体流出误差分析的MATLAB完整代码实现。

Part IV

附录

1 附录A：两相横流问题的MATLAB完整代码实现

```
% Fluid parameters
density_fluid = 1.0;         % Fluid density [kg/m^3]
U_max = 5;                   % Maximum centerline velocity [m/s]

% Particle parameters
radius = 0.01;               % Particle radius [m]
density_particle = 2.8;      % Particle density [kg/m^3]
mass = (4/3) * pi * (radius^3) * (density_particle - density_fluid);
   % Equivalent Particle mass [kg]
area = pi * (radius^2);      % Cross-sectional area [m^2]

% Drag coefficient
C_d = 0.50;                  % Dimensionless drag coefficient

% Gravity
g = 9.81;                    % Gravitational acceleration [m/s^2]

% Time parameters
dt = 0.01;                   % Time step size [s]
total_time = 1000;           % Total simulation time [s]
num_steps = round(total_time / dt); % Number of time steps
sample_rate = 4;

% Initial conditions
initial_position_1 = [2.5; 1.0];    % IC1: [x; y] [m]
initial_velocity_1 = [0.0; -1.0];   % IC1: [vx; vy] [m/s]
initial_positions_2 = [0.0, 0.40; 0.0, 0.0; 0.0, -0.40]; % IC2
    positions [m]
initial_velocity_2 = [1.6; 0.0];    % IC2 velocity [m/s]

% Fluid velocity profile
fluid_velocity = @(y) [U_max * (1 - y^2); 0];

% Drag force calculation
drag_force = @(v_particle, v_fluid) ...
    0.5 * C_d * density_fluid * norm(v_fluid - v_particle) * ...
    (v_fluid - v_particle) * area;
```

```matlab
% ===== Simulation =====
% IC1 Simulation
positions_1 = zeros(2, num_steps + 1);
positions_1(:, 1) = initial_position_1;
velocity = initial_velocity_1;

for t = 1:num_steps
    v_fluid = fluid_velocity(positions_1(2, t));
    drag = drag_force(velocity, v_fluid);
    gravity = [0; -mass * g];
    acceleration = (drag + gravity) / mass;
    velocity = velocity + acceleration * dt;
    positions_1(:, t + 1) = positions_1(:, t) + velocity * dt;

    % Boundary condition
    if abs(positions_1(2, t + 1)) > 1
        positions_1(2, t + 1) = sign(positions_1(2, t + 1)) * 1;
        velocity(2) = -0.5 * velocity(2); % Partial bounce
    end
end

% IC2 Simulation (3 particles)
trajectories_2 = cell(3, 1);
for i = 1:3
    pos_traj = zeros(2, num_steps + 1);
    pos_traj(:, 1) = initial_positions_2(i, :)';
    velocity = initial_velocity_2;

    for t = 1:num_steps
        v_fluid = fluid_velocity(pos_traj(2, t));
        drag = drag_force(velocity, v_fluid);
        gravity = [0; -mass * g];
        velocity = velocity + ((drag + gravity) / mass) * dt;
        pos_traj(:, t + 1) = pos_traj(:, t) + velocity * dt;

        % Boundary condition
        if abs(pos_traj(2, t + 1)) > 1
            pos_traj(2, t + 1) = sign(pos_traj(2, t + 1)) * 1;
            velocity(2) = -0.5 * velocity(2); % Partial bounce
        end
    end
    trajectories_2{i} = pos_traj;
end

% ===== Trajectory Plot =====
```

```
figure;
hold on;

% Plot IC1 trajectory as dots
scatter(positions_1(1,1:sample_rate:end), positions_1(2,1:sample_rate
    :end), ...
    20, 'filled', 'MarkerFaceColor', [0 0.447 0.741], 'DisplayName',
        'IC1: (2.5,0)');

% Plot IC2 trajectories as dots (3 particles)
colors = [1 0 0; 0 0.5 0; 0 0 1]; % Red, Green, Blue
labels = {'IC2: (0,0.4)', 'IC2: (0,0)', 'IC2: (0,-0.4)'};
for i = 1:3
    traj = trajectories_2{i};
    scatter(traj(1,1:sample_rate:end), traj(2,1:sample_rate:end), ...
        20, 'filled', 'MarkerFaceColor', colors(i,:), 'DisplayName',
            labels{i});
end

% Channel boundaries
yline(1, 'k--', 'LineWidth', 1.2);
yline(-1, 'k--', 'LineWidth', 1.2);

% Formatting
xlabel('x Position [m]', 'FontSize', 12);
ylabel('y Position [m]', 'FontSize', 12);
title('Particle Trajectories in Channel Flow', 'FontSize', 14);
legend('Location', 'northeast', 'FontSize', 10);
grid on;
axis equal;
xlim([0, 10]);
ylim([-1.2, 1.2]);
set(gca, 'FontSize', 10);
box on;
hold off;
```

2 附录B：颗粒轨迹的MATLAB完整代码实现

```
function state = ComputeTrajectory(dt, timesteps)
    % Initialize state matrix (2 rows x timesteps columns)
    state = zeros(2, timesteps);

    % Initial position (0,0)
    x = 0;
```

```
    y = 0;

    for k = 1:timesteps
        % Calculate velocity at current timestep
        vx = cos(6 * (k-1) * dt);  % t = (k-1)*dt
        vy = sin(6 * (k-1) * dt);

        % Update position using Euler integration
        x = x + vx * dt;
        y = y + vy * dt;

        % Store current state
        state(:, k) = [x; y];
    end
end

% Test script
dt = 0.005;            % Time step size (seconds)
timesteps = 2000;      % Number of time steps

% Compute trajectory
trajectory = ComputeTrajectory(dt, timesteps);

% Extract x and y coordinates
x = trajectory(1, :);
y = trajectory(2, :);

% Plot trajectory
figure;
plot(x, y, 'b-', 'LineWidth', 1.5);
xlabel('x position');
ylabel('y position');
title('Particle Trajectory: y vs x');
grid on;
axis equal;            % Equal scaling for x and y axes

% Mark starting point
hold on;
plot(0, 0, 'ro', 'MarkerSize', 8, 'MarkerFaceColor', 'r');
hold off;
legend('Trajectory', 'Start Point (0,0)');
```

3 附录C：水箱内液体流出的MATLAB完整代码实现

```matlab
% watertank.m - Simulation of Liquid Drainage from an Inverted Conical Tank
clear; clc; close all;

%% 1. Parameter Settings
r_h = 0.025;        % Radius of the bottom orifice (m)
g = 9.81;           % Gravitational acceleration (m/s^2)
y0 = 2.5;           % Initial liquid height (m)
t_end = 3000;       % Total simulation time (s)
dt = 0.1;           % Time step (s)

%% 2. Define the Differential Equation (Anonymous Function)
dydt = @(t,y) - (sqrt(2*g*y) * r_h^2) / (2 - 0.5*y)^2;

%% 3. Time Array Setup
tspan = 0:dt:t_end; % From 0 to 3000s with step size 0.1s

%% 4. Solve the Differential Equation Using ode45
% Set relative tolerance for improved accuracy
options = odeset('RelTol',1e-6);
[t, y] = ode45(dydt, tspan, y0, options);

% Convert output to column vectors
t = t(:);
y = y(:);

%% 5. Find When Liquid Height First Falls Below 0.1m
empty_index = find(y < 0.1, 1);
if isempty(empty_index)
    empty_index = length(t);
    warning('Liquid did not drain below 0.1m within simulation time');
else
    fprintf('Time when liquid height first falls below 0.1m: %.1f seconds\n', t(empty_index));
end

%% 6. Plot Liquid Height vs Time
figure;
plot(t(1:empty_index), y(1:empty_index), 'LineWidth', 2);
hold on;
plot(t(empty_index), y(empty_index), 'ro', 'MarkerSize', 8); % Mark drainage point
hold off;
```

```
% Plot formatting
xlabel('Time (s)', 'FontSize', 12);
ylabel('Liquid Height y (m)', 'FontSize', 12);
title('Liquid Height vs Time in Inverted Conical Tank', 'FontSize',
    14);
grid on;
set(gca, 'FontSize', 11);

% Add annotation
annotation('textbox', [0.6, 0.7, 0.2, 0.1], 'String', ...
    sprintf('Drainage time: %.1f s', t(empty_index)), ...
    'FitBoxToText', 'on', 'BackgroundColor', 'white');
```

4 附录C_1：理论y-t关系的MATLAB完整代码实现

```
% Define parameters
r_h = 0.025; % Radius of the hole at the bottom, in meters
y_0 = 2.5; % Initial height of the liquid, in meters

% Define the function t(y)
t_y = @(y) (1 / sqrt(2*9.81)/r_h^2) * (8 * (y_0^(1/2) - y.^(1/2)) -
    (4/3) * (y_0^(3/2) - y.^(3/2)) + (1/10) * (y_0^(5/2) - y.^(5/2)));

% Define the range of y, from 0 to 2.5 meters
y_values = linspace(0.01, 2.5, 10000); % From 0 to 2.5 meters

% Calculate the corresponding t values
t_values = t_y(y_values);

% Plot the graph
figure;
plot(t_values, y_values, 'LineWidth', 2);
xlabel('Time t (s)');
ylabel('Liquid Height y (m)');
title('Time as a Function of Liquid Height');
grid on;
```

5 附录C_2：误差分析的MATLAB完整代码实现

```
% Define parameters
r_h = 0.025; % Radius of the hole, in meters
y_0 = 2.5; % Initial height of the liquid, in meters
g = 9.81; % Acceleration due to gravity, in m/s^2
t_end = 3000; % Total simulation time, in seconds
```

```matlab
dt = 0.1; % Time step, in seconds

% Define the function t(y)
t_y = @(y) (1 / r_h^2/sqrt(2*9.81)) * (8 * (y_0^(1/2) - y.^(1/2)) -
    (4/3) * (y_0^(3/2) - y.^(3/2)) + (1/10) * (y_0^(5/2) - y.^(5/2)));

% Define the differential equation
dydt = @(t,y) - (sqrt(2*g*y) * r_h^2) / (2 - 0.5*y)^2;

% Set up the time array
tspan = 0:dt:t_end; % From 0 to 3000 seconds, with a step size of 0.1

% Solve the differential equation using ode45
options = odeset('RelTol',1e-6);
[t_ode, y_ode] = ode45(dydt, tspan, y_0, options);

% Calculate the theoretical values t(y)
y_values = linspace(0.001, y_0, 1000); % From 0 to initial height
t_values = t_y(y_values);

% Plot the graph
figure;
plot(t_ode, y_ode, 'b-', 'LineWidth', 2); % ode45 results
hold on;
plot(t_values, y_values, 'r--', 'LineWidth', 2); % Theoretical curve
hold off;

% Format the plot
xlabel('Time (s)', 'FontSize', 12);
ylabel('Liquid Height y (m)', 'FontSize', 12);
title('Comparison of Liquid Height vs Time', 'FontSize', 14);
legend('ode45 Simulation', 'Theoretical Curve', 'Location', 'best');
grid on;
set(gca, 'FontSize', 11);

% Calculate and plot the difference
figure;
diff_y = y_ode - interp1(t_values, y_values, t_ode);
plot(t_ode, diff_y, 'LineWidth', 2, 'Color', 'magenta');
xlabel('Time (s)', 'FontSize', 12);
ylabel('Difference in Liquid Height (m)', 'FontSize', 12);
title('Difference Between Simulation and Theory', 'FontSize', 14);
grid on;
set(gca, 'FontSize', 11);
```