SİVAS
BİLİM VE TEKNOLOJİ
ÜNİVERSİTESİ

Module Project

Computer Engineering Department

Contents

SİVAS
BİLİM VE TEKNOLOJİ
ÜNİVERSİTESİ

Project Development Process

01 Tasks

Physic Problems
Algorithm
Flowchart
C++ Codes

02 Graphs and Simuations

Matlab
Phyton

- Task Distribution
- Challenges Encountered
- Solitions
- Achievements and lesson learned

03 Preparing Report

Methods and approachs for taskes

04 Presenting

Presentation struction
Timing

SİVAS
BİLİM VE TEKNOLOJİ
ÜNİVERSİTESİ

C++ Codes (used in common with all codes)

Constants

```cpp
#include <iostream>
#include <cmath>
#include <cctype>

using namespace std;
const float g = 9.8;
const float PI = 3.14159;
const float G = 6.6743 * pow(10, -11);
```

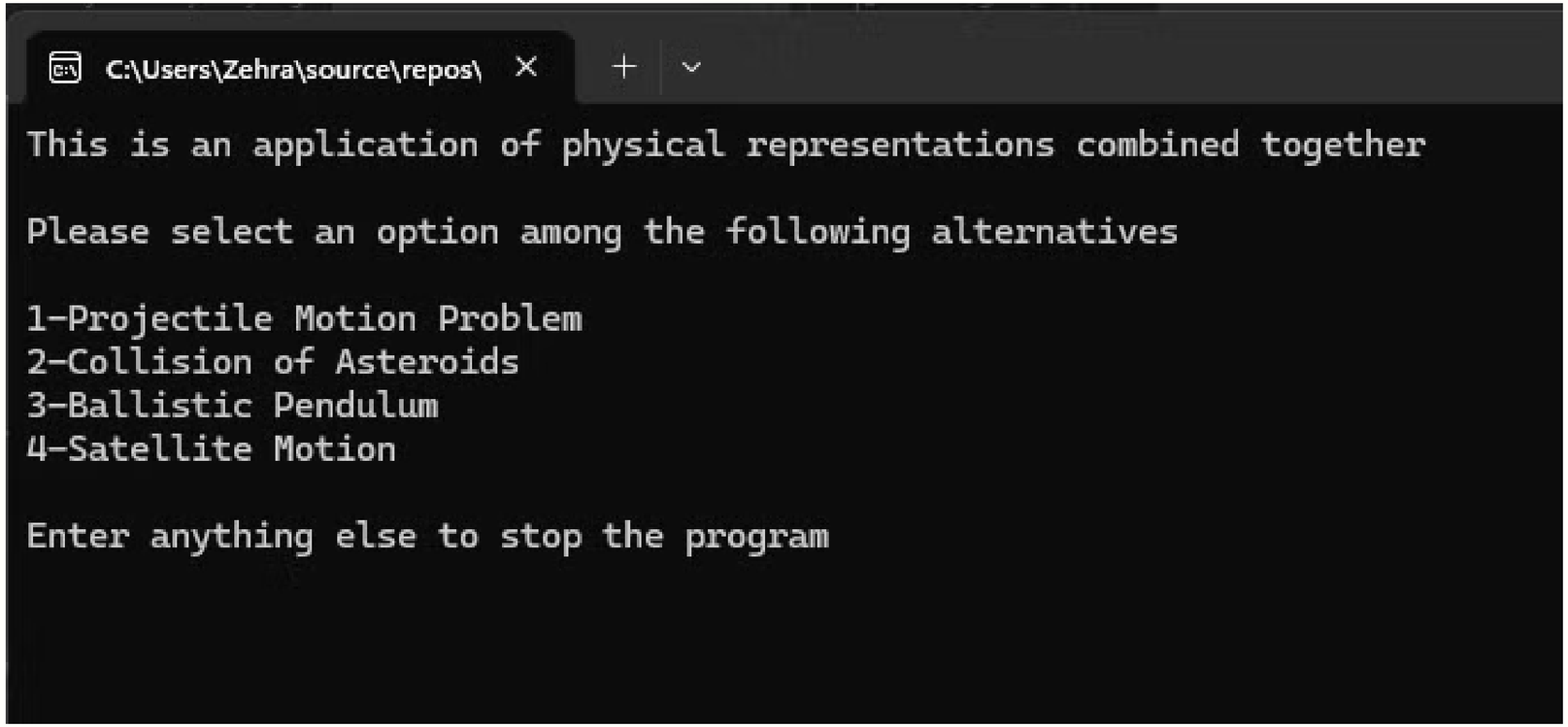
```cpp
void menu();

// Air resistance constants
const double air_density = 1.225; // Air density
const double Cd = 0.47; // Drag coefficient
```

Menu Code for Selecting Tasks

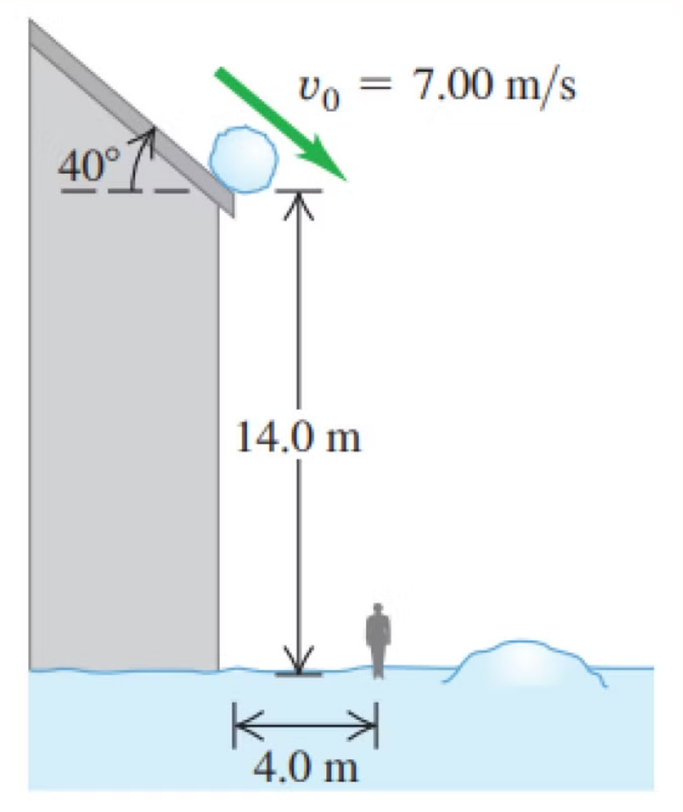
```cpp
//interactive main menu
void menu(){
    int ans;

    cout << "This is an application of physical representations combined together\n";
    cout << "\nPlease select an option among the following alternatives\n";
    cout << "\n1-Projectile Motion Problem";
    cout << "\n2-Collision of Asteroids";
    cout << "\n3-Ballistic Pendulum";
    cout << "\n4-Satellite Motion";
    cout << "\n\nEnter anything else to stop the program\n";
    cin >> ans;

    //options
    switch(ans){
    case 1:
        system("cls");
        motion();
        break;
    case 2:
        system("cls");
        collision();
        break;
    case 3:
        system("cls");
        pendulum();
        break;
    case 4:
        system("cls");
        satellite();
        break;
    default:
        system("cls");
        std::cout << "program has ended thank you for using";
        break;
    }

}

int main(){

    menu();

    return 0;
}
```

Console (used in common with all codes)

Menu in Console Application



Task 1: Projectile Motion

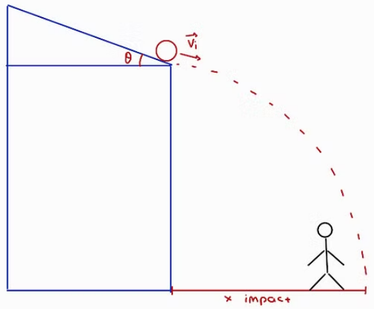


A snowball rolling from a roof with an initial speed and angle

- Calculate the path
- Air resistance by choice
- Hitting person on the road

Physics Formulas

Projectile motion



$$V_{initial} = V_0$$

$$V_x = V_0 \cdot \cos\theta \quad \text{(remains constant)}$$

$$V_y = -V_0 \cdot \sin\theta$$

$$t_{total} = \frac{-V_{y0} + \sqrt{V_{y0}^2 + 2gh}}{g} \quad \text{(total motion time)}$$

$$X = V_x \cdot t \quad \text{(horizontal motion)}$$

$$y = V_y \cdot t - \frac{1}{2} g t^2 \quad \text{(vertical motion)}$$

$$X_{impact} = V_x \cdot t_{total} \quad \text{(range)}$$

$$h_{initial} = 14$$

$$X_{moment} = V_x \cdot t_{moment}$$

$$y_{moment} = h_{initial} + V_y \cdot t_{moment} - \frac{1}{2} g t_{moment}^2$$

If it hits the person

$$\frac{X_{person}}{V_x} = t_{hit} \quad \text{(the time at ball hits the person)}$$

$$y_{hit} = h_{initial} + V_y \cdot t_{hit} - \frac{1}{2} g t_{hit}^2 \quad \text{(the height at which it hits)}$$

With friction

$$F_{drag} = \frac{1}{2} \cdot C_d \cdot \rho \cdot A \cdot V^2 \quad \text{(resistance force)}$$

$$V = \sqrt{V_x^2 + V_y^2}$$

(C_d = drag coefficient)

(ρ = air density)

(A = cross-sec area of object)

(V = velocity of the object)

$$a_x = -\frac{F_{drag}}{m} \cdot \frac{V_x}{V} \qquad a_y = -g - \frac{F_{drag}}{m} \cdot \frac{V_x}{V}$$

$$V_x = V_x + a_x \cdot \Delta t \qquad V_x = V_x + a_y \cdot \Delta y$$

$$X = x + V_x \cdot \Delta t \qquad y = y + V_y \cdot \Delta t$$

Algorithm and flowchart

Projectile motion

1) give the constants that we will use for the calculations
- g=9.8
- PI = 3.141592653589793238 46
- air_density = 1.225
- Cd(drag coefficient) = 0.47

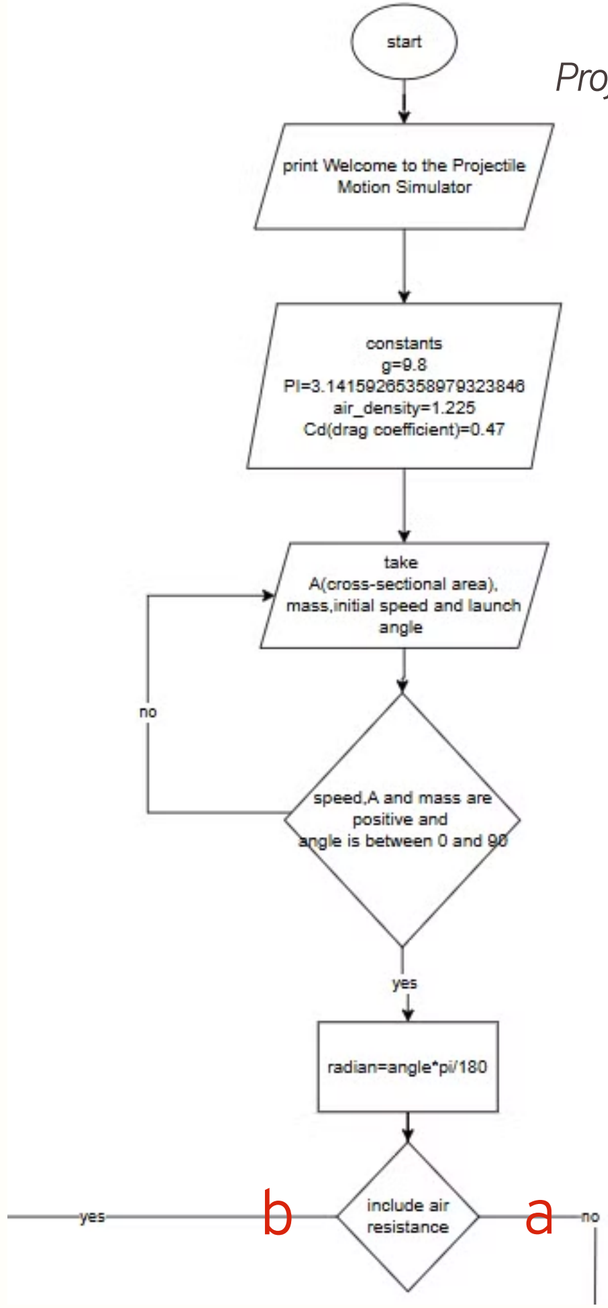
2) get the mass,cross-sectional area, initial speed and the angle from the user

3) check if the given values are true(angle must be between 0 and 90) (speed,mass and cross-sectional area must be positive)

4) If the values has no problem; continue to the next step. If there is problem; want the user to give accurate values

5) transform the given angle to radian with the equation rad=angle*pi/180

6) ask user rather s/he wants to include air resistance or not



Algorithm and flowchart

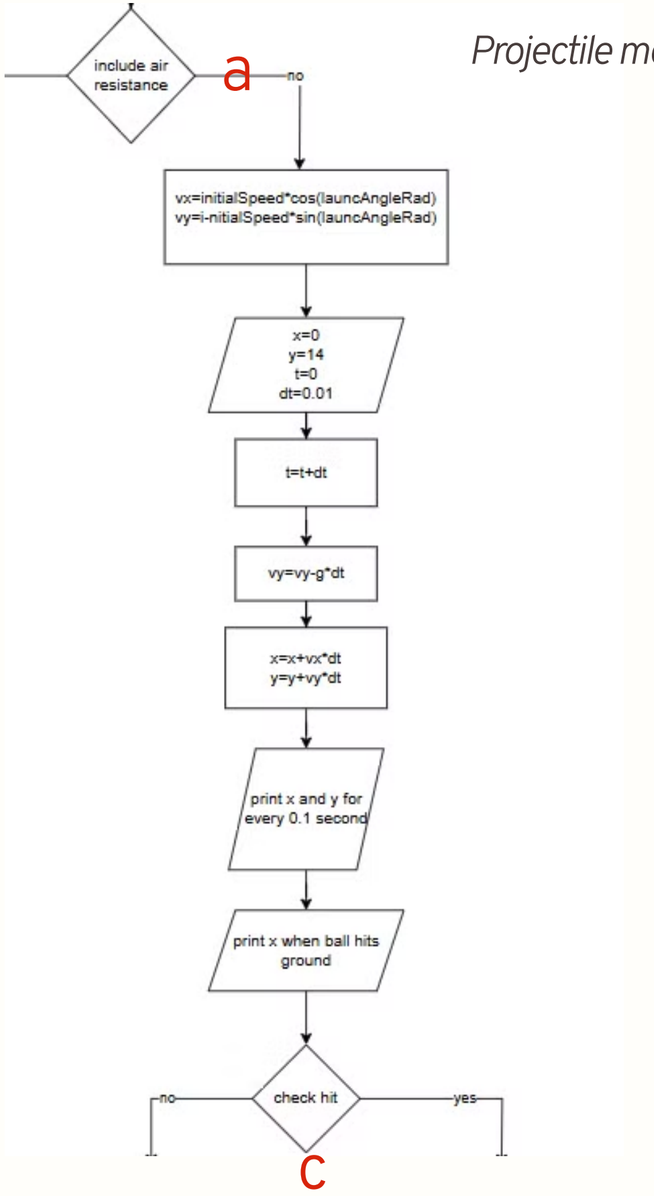
7) If the answer is no, calculate the speed's x and y components using the formulas
- vx = initialSpeed * cos(launchAngleRad);
- vy = -initialSpeed * sin(launchAngleRad

8) give the initial x value(0) and initial y value(14) of the ball
9) give the starting time (0) and we give dt(time step that we want) as 0.01
10) change time in every step
- t+=dt

11) calculate new vy values according to time
- vy -= g * dt

12) calculate the x and y coordinates of the ball due to time
- x += vx * dt;
- y += vy * dt

13) print out the x and y positions of the ball in every 0.1 second
14) print the last x position of the ball when it hits the ground
15) ask the user that if s/he wants to chech if the ball hits a person on the way or not



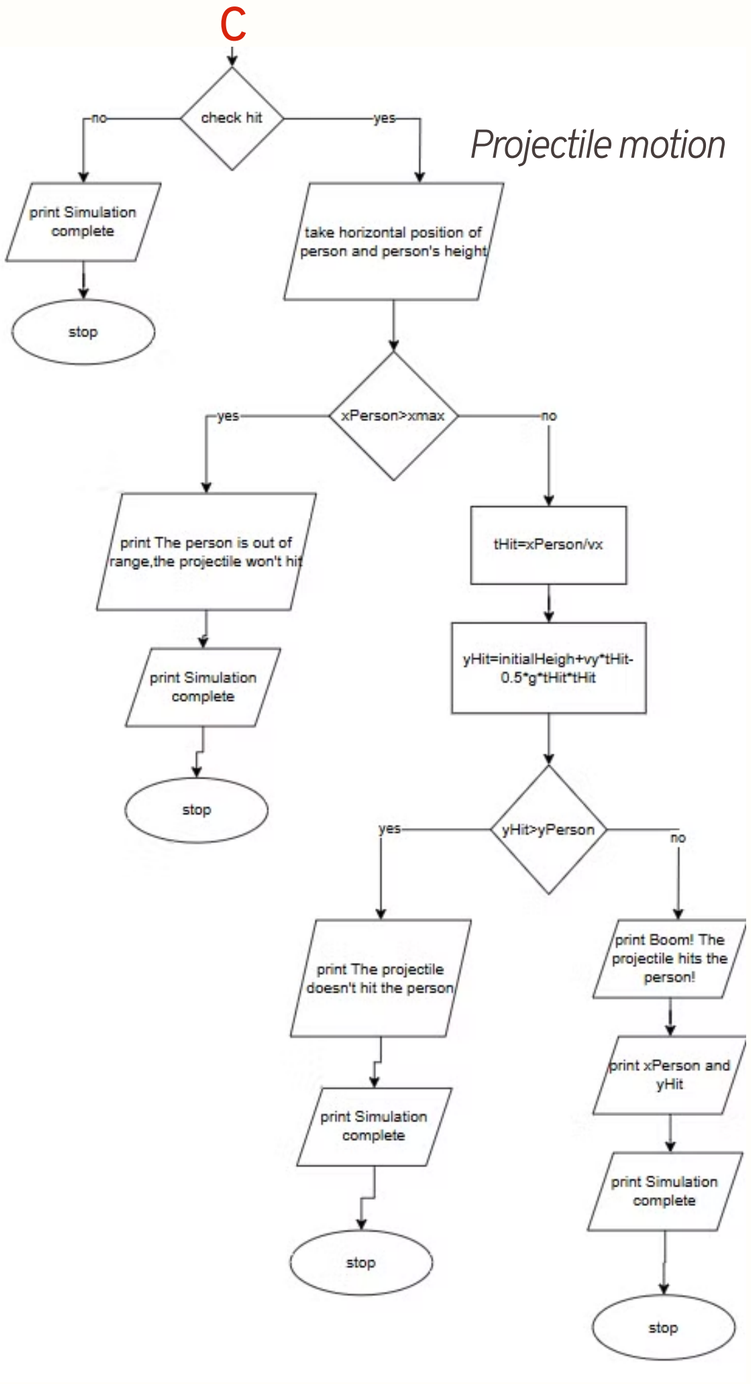
Projectile motion

Algorithm and flowchart

16) if the user wants to check the hit, get the x location off the person and height of the person
17) chech the persons horizontal position and if it's bigger than the x value hen the ball hits ground, the code prints out:
The person is out of range. The projectile won't hit them
18) calculate the hitting time
- tHit = xPerson/vx

19) calculate ball's vertical position at that time
- yHit = initialHeight + vy * tHit - 0.5 * g * tHit * tHit

20) if yHit is smaller or equal to the person's height; print
"Boom! The projectile hits the person! and print the x and y position's of hit
21) if yHit is bigger than person's height;print
"The projectile doesn't hit the person.
22)if user doesn't want to check hit; print
simulation comlete



Algorithm and flowchart

23) If the answer is yes to air resistance, calculate the speed's x and y components using the formulas

- vx = initialSpeed * cos(launchAngleRad)
- vy = -initialSpeed * sin(launchAngleRad),

24) give the initial x value(0) and initial y value(14) of the ball

25) give the starting time (0) and give dt(time step that we want) as 0.01

26) calculate air resistance

- resistance=0.5 * Cd * A * air_density * v * v

27) calculate the chaging values of acceleration for x, speed for x and horizontal position of the ball due to time

- a_x = -resistance / m * (vx / v)
- vx += a_x * dt
- x += vx * dt

28) calculate the chaging values of acceleration for y, speed for y and verical position of the ball due to time

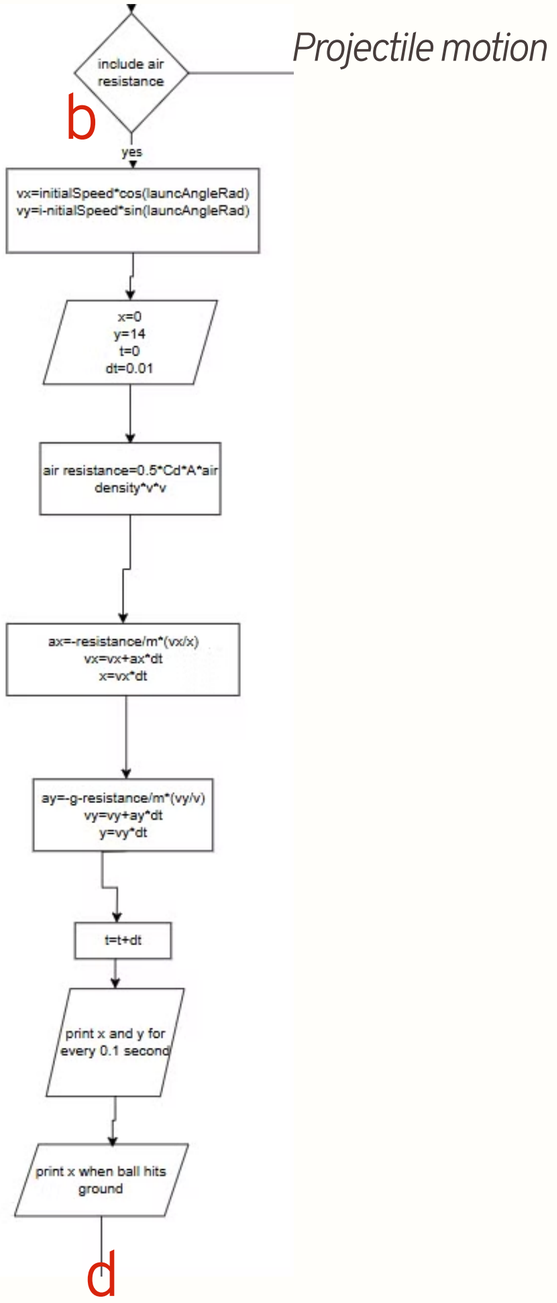
- a_y = -g - resistance / m * (vy / v)
- vy += a_y * dt
- y += vy * dt

29) change time in every step

- t+=dt

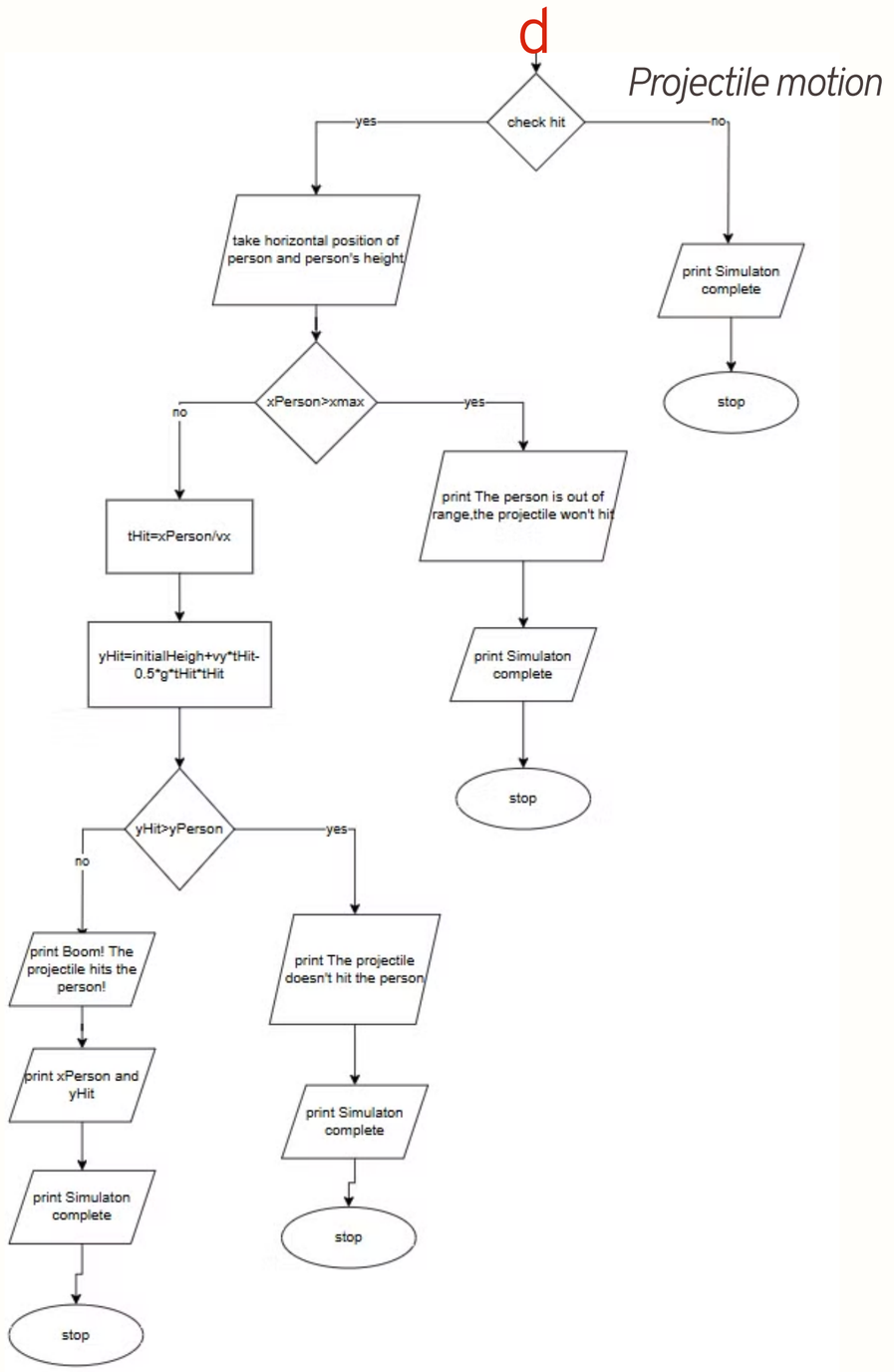
30) print out the x and y positions of the ball in every 0.1 second

31) print the last x position of the ball when it hits the ground



Algorithm and flowchart

32) ask the user that if s/he wants to chech if the ball hits a person on the way or not
33) if the user wants to check the hit, get the x location off the person and height of the person
34) chech the persons horizontal position and if it's bigger than the x value hen the ball hits ground, the code prints out:
The person is out of range. The projectile won't hit them
35) calculate hitting time
tHit = xPerson / vx
36) calculate ball's vertical position at that time
yHit = initialHeight + vy * tHit - 0.5 * g * tHit * tHit
37) if yHit is smaller or equal to the person's height; print
"Boom! The projectile hits the person! and print the x and y position's of hit
38) if yHit is bigger than person's height;print
"The projectile doesn't hit the person.
39) if user doesn't want to check hit; print
simulation comlete



C++ Codes

functions


```cpp
// Function to compute air resistance
double resistanceForce(double v) {
    return 0.5 * Cd * A * air_density * v * v;
}
```

```cpp
// Function to validate and get positive inputs
double getPositiveInput(string prompt) {
    double value;
    do {
        cout << prompt;
        cin >> value;
        if (value <= 0) {
            cout << "Oops! The value must be positive. Please try again." << endl;
        }
    } while (value <= 0); // Repeat until we get a positive value
    return value;
}
```

```cpp
void checkHitNoAirResistance(double impactX, double initialHeight, double initialSpeed, double vxInitial, double vyInitial) {
    string hitCheck;
    do {
        cout << "Do you want to check if the projectile hits a person due to their location? (yes/no): ";
        cin >> hitCheck;
        if (hitCheck != "yes" && hitCheck != "no") {
            cout << "Please type just 'yes' or 'no'!!!" << endl;
        }
    } while (hitCheck != "yes" && hitCheck != "no");// to get answer just yes or no

    if (hitCheck == "yes") {
        double xPerson = getPositiveInput("Enter the horizontal position of the person (m): ");
        double yPerson = getPositiveInput("Enter the height of the person (m): ");

        // Check if the person is within the horizontal range
        if (xPerson > impactX) {
            cout << "The person is out of range. The projectile won't hit them." << endl;
        }
        else {
            double tHit = xPerson / vxInitial; // Time at which projectile would hit person's x position
            double yHit = initialHeight + vyInitial * tHit - 0.5 * g * tHit * tHit;

            if (yHit <= yPerson) {
                cout << "Boom! The projectile hits the person!" << endl;
                cout << "Position at impact: X = " << xPerson << " m, Y = " << yHit << " m" << endl;
            }
            else {
                cout << "The projectile doesn't hit the person." << endl;
            }
        }
    }
}
```

C++ Codes

functions

```cpp
// Function to check if the projectile hits a person (with air resistance)
void checkHitWithAirResistance(double initialHeight, double launchAngleRad, double impactX, double vx, double vy, double A, double m) {
    string hitCheck;
    do {
        cout << "Do you want to check if the projectile hits a person due to their location? (yes/no): ";
        cin >> hitCheck;
        if (hitCheck != "yes" && hitCheck != "no") {
            cout << "Please type just 'yes' or 'no'!!!" << endl;
        }
    } while (hitCheck != "yes" && hitCheck != "no");

    if (hitCheck == "yes") {
        double xPerson = getPositiveInput("Please enter the horizontal position of the person (in m): ");
        double yPerson = getPositiveInput("Please enter the height of the person (in m): ");

        // Check if the person is within the horizontal range
        if (xPerson > impactX) {
            cout << "The person is out of range. The projectile won't hit them." << endl;
        }
        else {  // Simulate the projectile's motion and check if it hits the person
            double x = 0.0;
            double y = initialHeight;
            double vxCurr = vx; // Current horizontal velocity
            double vyCurr = vy; // Current vertical velocity
            double t = 0.0;
            double dt = 0.0000001;
            double a_x;
            double a_y;

            while (x <= xPerson && y!= 0) {
                a_x = -(resistanceForce(vx, A) / m);
                vx += a_x * dt;
                x += vx * dt;
                a_y -= g +resistanceForce(vy, A) / m;
                vy += a_y * dt;
                y += vy * dt;
                t += dt;
            }

            // If the horizontal position of the projectile is at the person's position
            if (y <= yPerson) {
                cout << "Boom! The projectile hits the person!" << endl;
                cout << "Position at impact: X = " << xPerson << " m, Y = " << y << " m" << endl;
                return; // Exit the function once the projectile hits the person
            }

            // If the projectile does not hit the person
            cout << "The projectile doesn't hit the person." << endl;
        }
    }
}
```

C++ Codes

functions

```cpp
// Simulate projectile motion without air resistance
void simulateNoAirResistance(double initialSpeed, double launchAngleRad, double initialHeight) {
    double vxInitial = initialSpeed * cos(launchAngleRad); // Horizontal velocity
    double vyInitial = -initialSpeed * sin(launchAngleRad); // Vertical velocity (negative because it is downward)
    double vy = vyInitial;

    double x = 0.0;
    double y = initialHeight; // Initial height
    double dt = 0.00001; // Time step
    double t = 0.0;

    cout << "\nTime (s)\tX (m)\tY (m)" << endl;

    // Simulate motion until projectile hits the ground
    while (initialHeight + vyInitial * t - 0.5 * g * t * t >= 0) {
        x += vxInitial * dt;  // Horizontal movement
        vy -= g * dt;  // Gravity's effect on vertical velocity()
        y += vyInitial * dt - 0.5 * g * dt * dt;  // Vertical movement
        t += dt;

        if ((int)(t * 100000) % 10000 == 0) {
            cout << floor(t * 10) / 10 << "\t\t" << x << "\t" << initialHeight + vyInitial * t - 0.5 * g * t * t << endl;
        }
    }

    cout << "\nFinal horizontal position at impact: x = " << x << " meters" << endl;
    checkHitNoAirResistance(x,initialHeight, initialSpeed, vxInitial, vyInitial);
}
```

C++ Codes

functions

```cpp
// Simulate projectile motion with air resistance
void simulateWithAirResistance(double initialSpeed, double launchAngleRad, double A, double m, double initialHeight) {
    double vxInitial = initialSpeed * cos(launchAngleRad); // Horizontal velocity
    double vyInitial = -initialSpeed * sin(launchAngleRad); // Vertical velocity (negative because it is downward)
    double vx = vxInitial;
    double vy = vyInitial;
    double x = 0.0;
    double y = initialHeight; // Initial height
    double dt = 0.0000001; // Time step
    double t = 0.0;
    double a_x;
    double a_y;

    cout << "\nTime (s)\tX (m)\tY (m)" << endl;

    // Simulate motion until projectile hits the ground
    while (y >= 0) {

        // Update horizontal acceleration due to air resistance
        a_x = -(resistanceForce(vx, A) / m);
        vx += a_x * dt;
        x += vx * dt;

        // Update vertical acceleration due to gravity and air resistance
        a_y = - g + resistanceForce(vy, A) / m;
        vy += a_y * dt;
        y += vy * dt;

        t += dt;

        if ((int)(t * 10000000) % 1000000 == 0) {
            cout << floor(t * 10) / 10 << "\t\t" << x << "\t" << y << endl;
        }
    }

    cout << "\nFinal horizontal position at impact: x = " << x << " meters" << endl;
    checkHitWithAirResistance(initialHeight, launchAngleRad, x, vxInitial, vyInitial, A, m);
}
```

C++ Codes

main

```cpp
void motion() {
    cout << "============================================================" << endl;
    cout << "        Welcome to the Projectile Motion Simulator          " << endl;
    cout << "============================================================" << endl;

    cout << "\nthe aim of this application is to simulate motion of an object which is released from height of 14 meters\n";
    cout << "with a specific angle and to check if it hits a person which is at the platform\n";
    cout << "now to get started to calculations\n";

    string ans;
    do {

        // Get user input
        double initialSpeed = getPositiveInput("Please enter the initial speed (in m/s): ");
        double initialHeight = getPositiveInput("Please enter the initial Height (in m): ");
        double launchAngle = getValidAngle("Please enter the launch angle (in degrees): ");
        double launchAngleRad = launchAngle * PI / 180.0; // Convert degrees to radians

        // Ask user if air resistance should be included
        string includeAirResistance;
        do {
            cout << "Do you want to include air resistance? (yes/no): ";
            cin >> includeAirResistance;
            if (includeAirResistance != "yes" && includeAirResistance != "no") {
                cout << "Please type just 'yes' or 'no'!!!" << endl;
            }
        } while (includeAirResistance != "yes" && includeAirResistance != "no");

        if (includeAirResistance == "yes") {
            double A = getPositiveInput("Please enter the cross-sectional area (A) (in m^2): ");
            double m = getPositiveInput("Enter the mass of object (in kg): ");

            simulateWithAirResistance(initialSpeed, launchAngleRad, A, m, initialHeight);
        }
        else {
            simulateNoAirResistance(initialSpeed, launchAngleRad, initialHeight);
        }
        cout << "\nnow if you want to make another calculation please enter 'yes' or 'no' to go back to menu\n";
        cin >> ans;
        for (auto& c : ans) c = tolower(c);

        while (ans != "yes" && ans != "no") {
            cout << "please enter 'yes' or 'no'\n";
            std::cin >> ans;
            for (auto& c : ans) c = tolower(c);
        }

        system("cls");

    } while (ans == "yes");

    menu();
}
```

Console

Projectile Motion Without Air Resistance

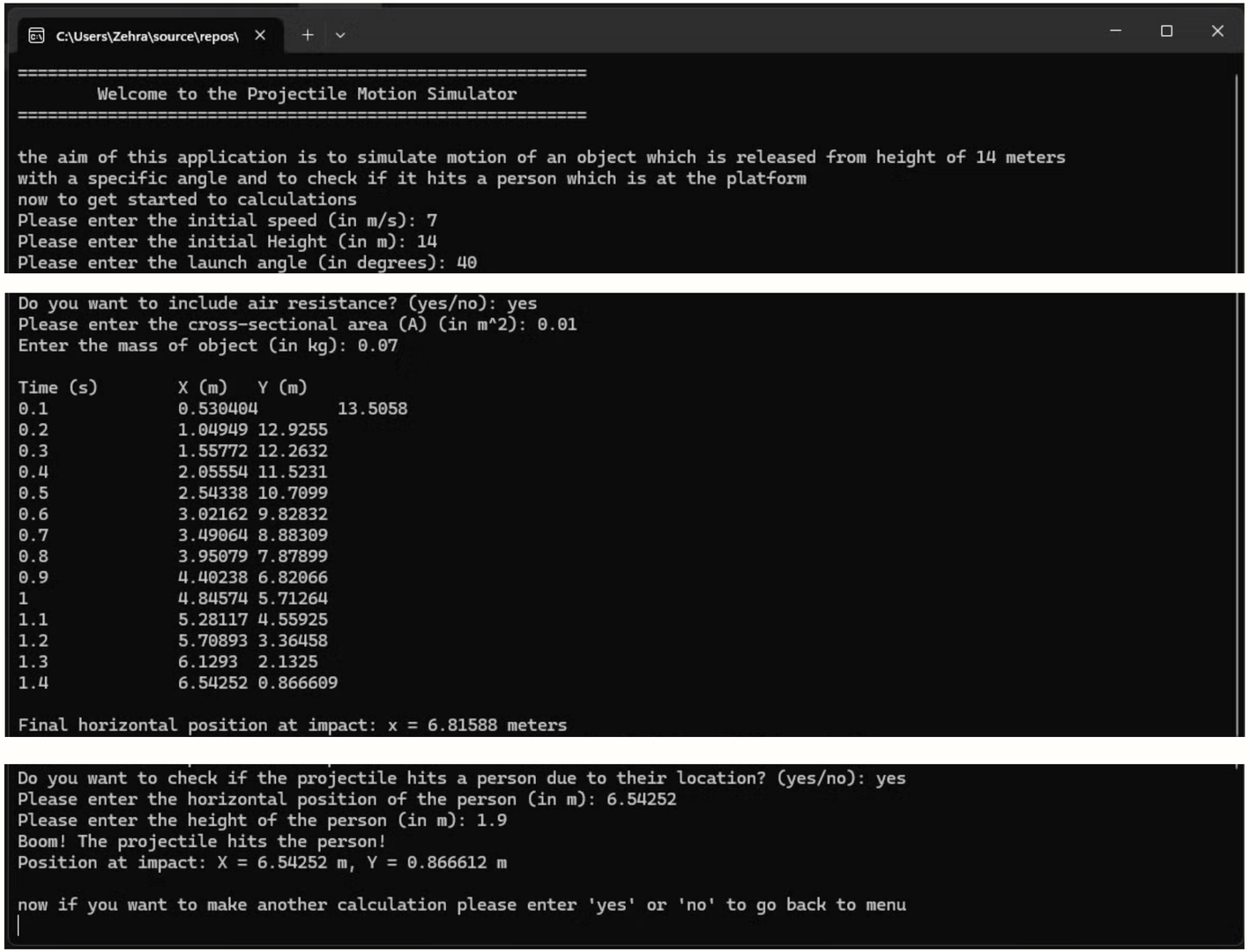
```
C:\Users\Zehra\source\repos\
=====================================================
        Welcome to the Projectile Motion Simulator
=====================================================

the aim of this application is to simulate motion of an object which is released from height of 14 meters
with a specific angle and to check if it hits a person which is at the platform
now to get started to calculations
Please enter the initial speed (in m/s): 7
Please enter the initial Height (in m): 14
Please enter the launch angle (in degrees): 40
```

```
Do you want to include air resistance? (yes/no): no

Time (s)        X (m)   Y (m)
0.1             0.536285        13.501
0.2             1.07246 12.9041
0.3             1.60869 12.2091
0.4             2.14493 11.4162
0.5             2.68116 10.5252
0.6             3.21744 9.53619
0.7             3.75367 8.44923
0.8             4.2899  7.26427
0.9             4.82614 5.98131
1               5.36237 4.60035
1.1             5.8986  3.12139
1.2             6.43483 1.54442

Final horizontal position at impact: x = 6.93042 meters
```

```
Do you want to check if the projectile hits a person due to their location? (yes/no): yes
Enter the horizontal position of the person (in m): 6.43483
Enter the height of the person (in m): 1.9
Boom! The projectile hits the person!
Position at impact: X = 6.43483 m, Y = 1.54442 m

now if you want to make another calculation please enter 'yes' or 'no' to go back to menu
```

Console

Projectile Motion With Air Resistance



Graphs and Simulations

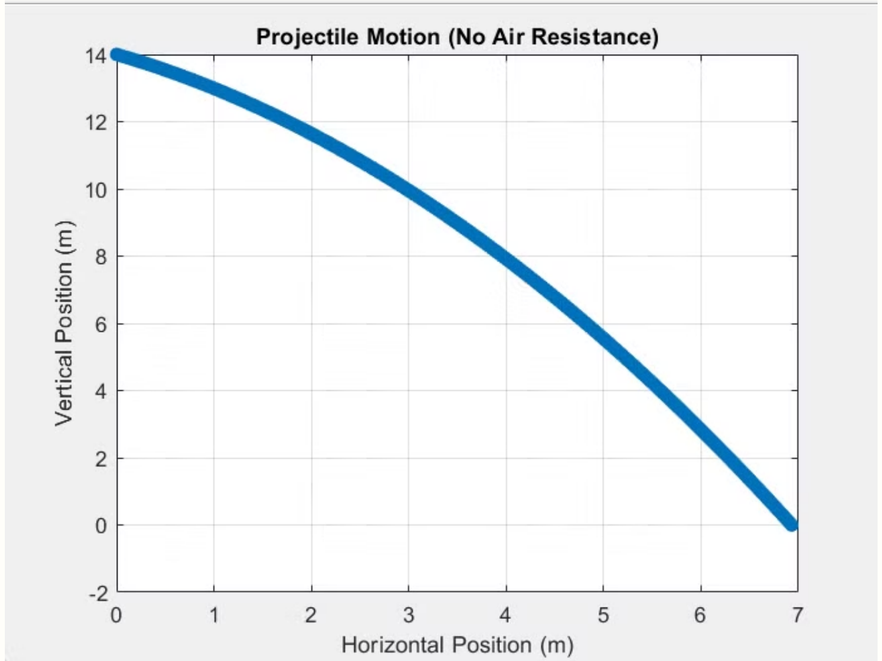
Projectile Motion Without Air Resistance

```
function projectileMotionSimulator()
    % Constants
    g = 9.8; % Gravitational acceleration (m/s^2)
    PI = 3.14159; % Pi value
    initialHeight = 14; % Initial height (meters)
    initialSpeed = 7;
    launchAngle=40;
    % Convert angle to radians
    launchAngleRad = launchAngle * PI / 180;

    % Initial velocity components
    vxInitial = initialSpeed * cos(launchAngleRad); % Horizontal velocity component
    vyInitial = -initialSpeed * sin(launchAngleRad); % Vertical velocity component

    % Time and time step
    t = 0; % Initial time
    dt = 0.00001; % Time step (seconds)

    % Positions
    x = 0; % Horizontal position
    y = initialHeight; % Vertical position

    % Store time, x, and y data for plotting
    time = [];
```

```
    % Store time, x, and y data for plotting
    time = [];
    posX = [];
    posY = [];

    % Simulate motion
    while y >= 0
        % Update horizontal and vertical positions
        x = x + vxInitial * dt;
        y = y + vyInitial * dt - 0.5 * g * dt^2;

        % Update vertical velocity
        vyInitial = vyInitial - g * dt;

        % Record time, x, and y data
        t = t + dt;
        time = [time, t];
        posX = [posX, x];
        posY = [posY, y];
    end

    % Print results
    disp(['Final horizontal position at impact: ', num2str(x), ' meters']);
```

Graphs and Simulations

Projectile Motion Without Air Resistance




```
    % Plot graph
    figure;
    plot(posX, posY, '-o');
    title('Projectile Motion (No Air Resistance)');
    xlabel('Horizontal Position (m)');
    ylabel('Vertical Position (m)');
    grid on;
end
```

Graphs and Simulations

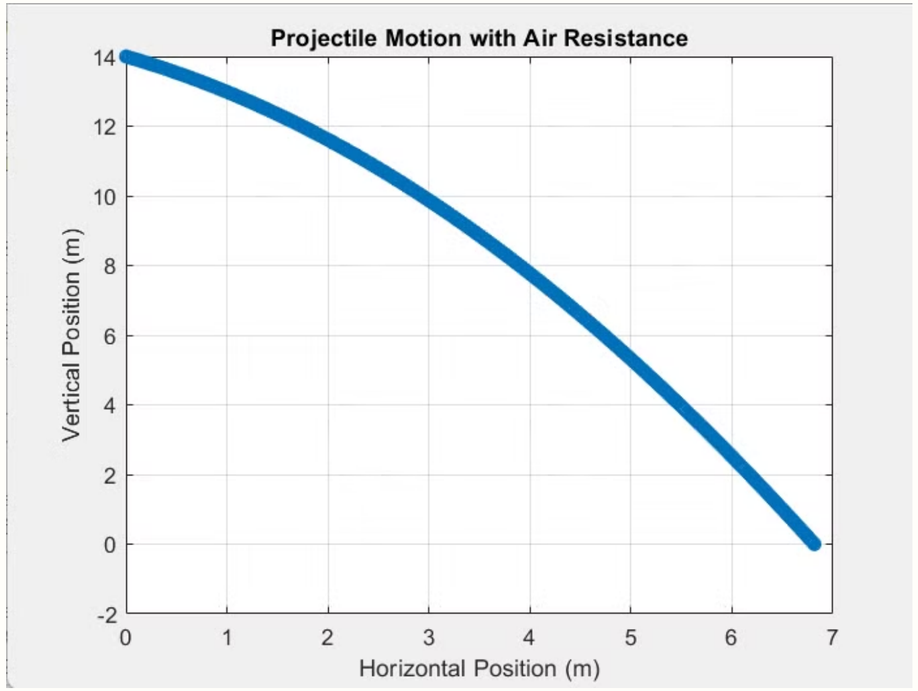
Projectile Motion With Air Resistance

```
function simulateWithAirResistance()
    % Constants
    g = 9.8; % Gravitational acceleration (m/s^2)
    airDensity = 1.225; % Air density (kg/m^3)
    Cd = 0.47; % Drag coefficient
    PI = 3.14159; % Pi value

    % Fixed input values
    initialSpeed = 7; % Initial speed (m/s)
    launchAngle = 40; % Launch angle (degrees)
    A = 0.001; % Cross-sectional area (m^2)
    m = 0.007; % Mass (kg)
    initialHeight = 14; % Initial height (m)

    % Convert angle to radians
    launchAngleRad = launchAngle * PI / 180;

    % Initial velocity components
    vx = initialSpeed * cos(launchAngleRad); % Horizontal velocity component
    vy = -initialSpeed * sin(launchAngleRad); % Vertical velocity component

    % Time and time step
    t = 0; % Initial time
    dt = 0.0001; % Time step (seconds)
```

```
    % Positions
    x = 0; % Horizontal position
    y = initialHeight; % Vertical position

    % Store time, x, and y data for plotting
    time = [];
    posX = [];
    posY = [];

    % Simulate motion
    while y >= 0
        % Calculate air resistance
        dragX = 0.5 * Cd * airDensity * A * vx^2 * sign(vx);
        dragY = 0.5 * Cd * airDensity * A * vy^2 * sign(vy);

        % Horizontal and vertical accelerations
        ax = -dragX / m;
        ay = -g - (dragY / m);

        % Update velocities
        vx = vx + ax * dt;
        vy = vy + ay * dt;
```

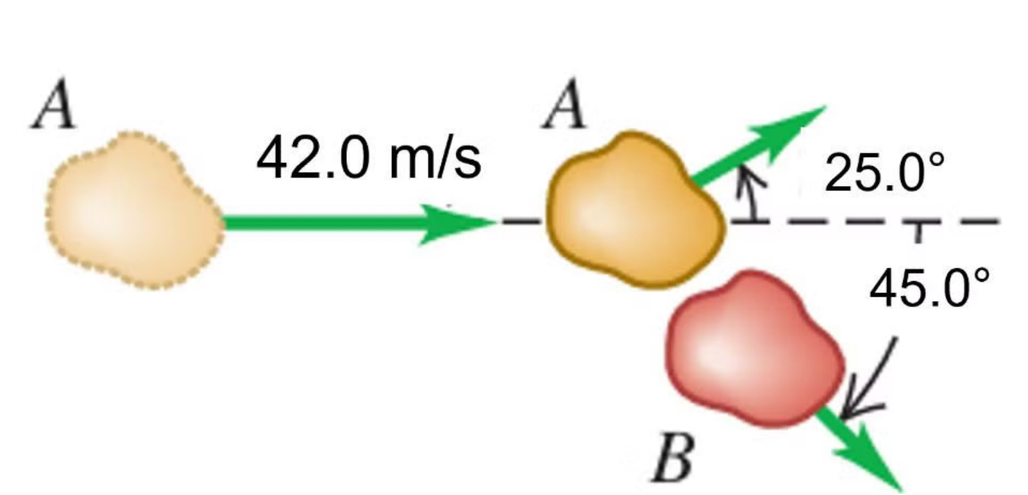
Graphs and Simulations

Projectile Motion With Air Resistance




```
        % Update positions
        x = x + vx * dt;
        y = y + vy * dt;

        % Update time
        t = t + dt;
        time = [time, t];
        posX = [posX, x];
        posY = [posY, y];
    end

    % Print results
    disp(['Final horizontal position: ', num2str(x), ' meters']);
    disp(['Final vertical position: ', num2str(y), ' meters']);
    disp(['Total flight time: ', num2str(t), ' seconds']);

    % Plot graph
    figure;
    plot(posX, posY, '-o');
    title('Projectile Motion with Air Resistance');
    xlabel('Horizontal Position (m)');
    ylabel('Vertical Position (m)');
    grid on;
end
```

Task 2: Collision of Asteroids



One asteroid with initial speed and angle hits the other one which is at rest

- Calculating final speeds
- Calculating final angels
- Calculating energy loss

Physics Formulas

Collision of Asteroids

$$\sum P_x + \sum P_y = P_i$$

$$P_x = P_i$$

$$P_y = 0$$

$$P_y = m\vec{V_1}\sin(\theta_1) - m\vec{V_2}\sin(\theta_2) = 0$$

$$\vec{V_1}\sin(\theta_1) = \vec{V_2}\sin(\theta_2)$$

$$\vec{V_2} = \vec{V_1}\frac{\sin(\theta_1)}{\sin(\theta_2)}$$

$$P_x = m\vec{V_1}\cos(\theta_1) - m\vec{V_2}\cos(\theta_2) = m\vec{V_i}$$

$$\vec{V_1}\cos(\theta_1) + \vec{V_2}\cos(\theta_2) = \vec{V_i}$$

$$\vec{V_1}\cos(\theta_1) + \frac{\vec{V_1}\cos(\theta_2)\cdot\sin(\theta_1)}{\sin(\theta_2)} = \vec{V_i}$$

$$\vec{V_1}\left(\frac{\cos(\theta_1)\sin(\theta_2) + \cos(\theta_2)\cdot\sin(\theta_1)}{\sin(\theta_2)}\right) = \vec{V_i}$$

Continious physics formulas:

$$\vec{V_1}\left(\frac{\sin(\theta_1+\theta_2)}{\sin(\theta_2)}\right) = \vec{V_i}$$

$$\vec{V_1} = \frac{\vec{V_i}\cdot\sin(\theta_2)}{\sin(\theta_1+\theta_2)}$$

$$\vec{V_2} = \frac{\vec{V_i}\cdot\sin(\theta_2)\cdot\sin(\theta_1)}{\sin(\theta_1+\theta_2)\sin(\theta_2)}$$

$$\vec{V_2} = \frac{\vec{V_i}\cdot\sin(\theta_1)}{\sin(\theta_1+\theta_2)}$$

Fraction of energy loss $= 100 \times \frac{E_i - E_f}{E_i}$

$$100 \cdot \frac{\frac{1}{2}m(\vec{V_i})^2 - \frac{1}{2}m(\vec{V_1})^2}{\frac{1}{2}m(\vec{V_i})^2}$$

$$100 \cdot \frac{(\vec{V_i})^2 - (\vec{V_1})^2}{(\vec{V_i})^2} \rightarrow \text{Fraction of energy loss}$$

Algorithm and Flowchart

1- Input:

- Initial speed of Asteroid A (vA_initial).
- Deflection angle of Asteroid A (θA).
- Deflection angle of Asteroid B (θB).

Process:

2- Start

3- Prompt the user to input:

- Initial speed of Asteroid A (vA_initial).
- Deflection angle of Asteroid A (θA) in degrees.
- Deflection angle of Asteroid B (θB) in degrees.

4- Convert the deflection angles (θA and θB) from degrees to radians.

- vA_final=(vA_initial*cos(θA))/(cos(θA)+sin(θB))
- vB_final = vA_final*(sin(θB)/cos(θA))

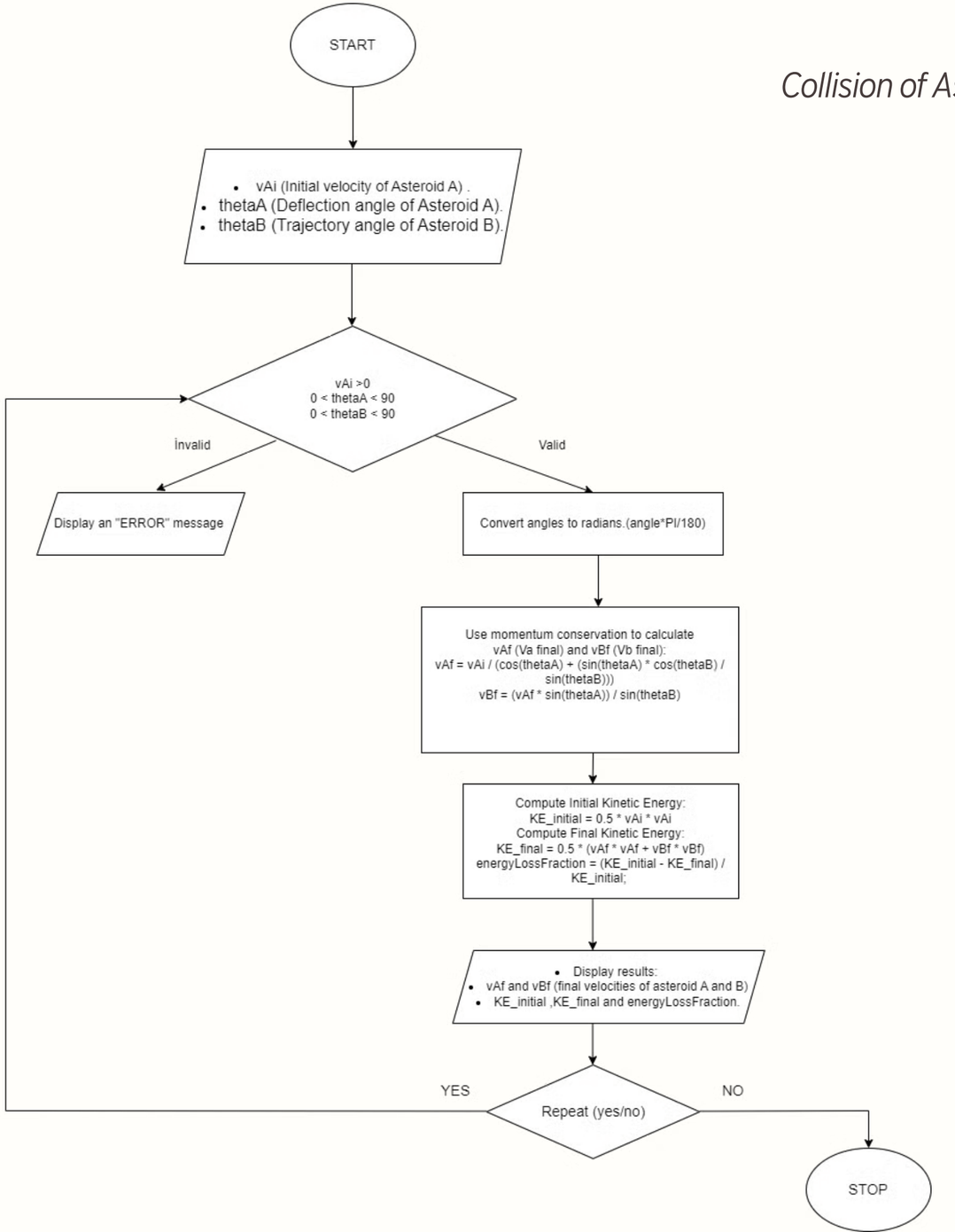
5- Calculate the kinetic energy:

- Initial Kinetic Energy (KE_initial) =0,5 * m* (vA_initial^2)
- Final Kinetic Energy (KE_final) = 0,5 *m * ((vA_final^2) + (vB_final^2))
- Energy Loss (KE_loss) = KE_initial -KE_final

6- Display:

- Final velocities of Asteroids A and B .
- Initial and final kinetic energies.
- Kinetic energy loss

7- End.



C++ Codes

functions

```cpp
// Function to calculate final velocities and kinetic energy loss
void calculateVelocities(double vAi, double thetaA, double thetaB, double& vAf, double& vBf, double& energyLossFraction) {
    // Convert angles to radians
    thetaA = thetaA * PI / 180.0;
    thetaB = thetaB * PI / 180.0;

    // Calculate final velocities
    vAf = vAi * (sin(thetaB) / sin(thetaA + thetaB));
    vBf = vAf * (sin(thetaA)) / sin(thetaB);

    // Kinetic energy calculations
    double KE_initial = 0.5 * vAi * vAi;  // Initial kinetic energy of asteroid A
    double KE_final = 0.5 * (vAf * vAf);  // Final kinetic energy of both asteroids
    energyLossFraction = (KE_initial - KE_final) / KE_initial;  // Fraction of energy lost
}
```

```cpp
// Function to print trajectory data
void printTrajectories(double vAf, double thetaA, double vBf, double thetaB) {
    // Convert angles to radians
    thetaA = thetaA * PI / 180.0;
    thetaB = thetaB * PI / 180.0;

    // Compute x and y components of velocities
    double xA = vAf * cos(thetaA);
    double yA = vAf * sin(thetaA);
    double xB = vBf * cos(thetaB);
    double yB = vBf * sin(thetaB);

    // Display trajectory data
    cout << "Trajectory Data (x, y):" << std::endl;
    cout << "Asteroid A: (" << xA << ", " << yA << ")" << std::endl;
    cout << "Asteroid B: (" << xB << ", " << yB << ")" << std::endl;
}
```

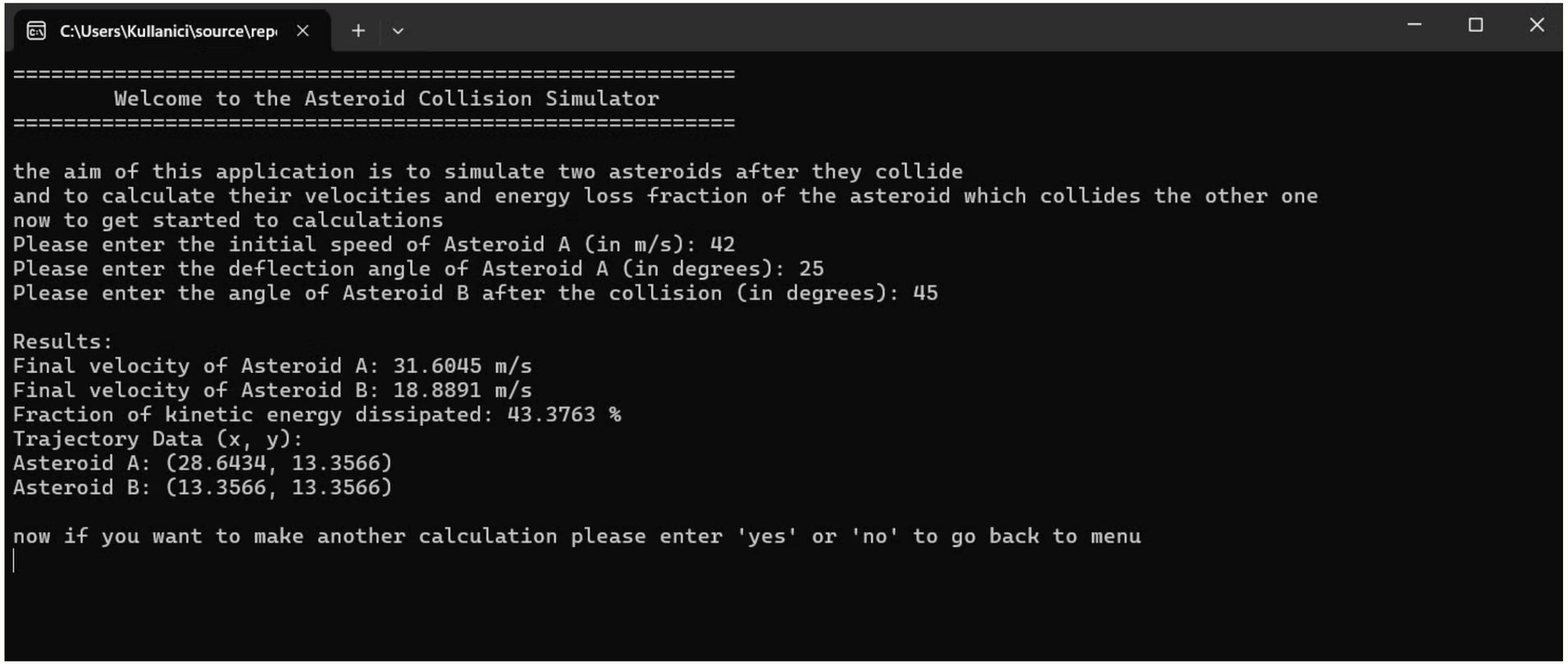
C++ Codes

main

```cpp
void collision() {

    cout << "=========================================================" << endl;
    cout << "        Welcome to the Asteroid Collision Simulator       " << endl;
    cout << "=========================================================" << endl;

    cout << "\nthe aim of this application is to simulate two asteroids after they collide\n";
    cout << "and to calculate their velocities and energy loss fraction of the asteroid which collides the other one\n";
    cout << "now to get started to calculations\n";

    string repeat;  // Initialize repeat character
    do {
        // Variables
        double vAi, thetaA, thetaB, vAf, vBf, energyLossFraction;

        // Input from user
        vAi = getPositiveInput("Please enter the initial speed of Asteroid A (in m/s): ");
        thetaA = getValidAngle("Please enter the deflection angle of Asteroid A (in degrees): ");
        thetaB = getValidAngle("Please enter the angle of Asteroid B after the collision (in degrees): ");

        // Perform calculations
        calculateVelocities(vAi, thetaA, thetaB, vAf, vBf, energyLossFraction);

        // Display results
        cout << "\nResults:" << std::endl;
        cout << "Final velocity of Asteroid A: " << vAf << " m/s" << std::endl;
        cout << "Final velocity of Asteroid B: " << vBf << " m/s" << std::endl;
        cout << "Fraction of kinetic energy dissipated: " << energyLossFraction * 100 << " %" << std::endl;

        // Print trajectory data
        printTrajectories(vAf, thetaA, vBf, thetaB);

        // Repeat the operation?
        cout << "\nnow if you want to make another calculation please enter 'yes' or 'no' to go back to menu\n";
        cin >> repeat;
        for (auto& c : repeat) c = tolower(c);
        system("cls");
        while (repeat != "yes" && repeat != "no") {
            cout << "please enter 'yes' or 'no'\n";
            std::cin >> repeat;
            for (auto& c : repeat) c = tolower(c);
        }

    } while (repeat == "yes");
    menu();

}
```

Console

Collision of asteroids



Graphs and Simulations

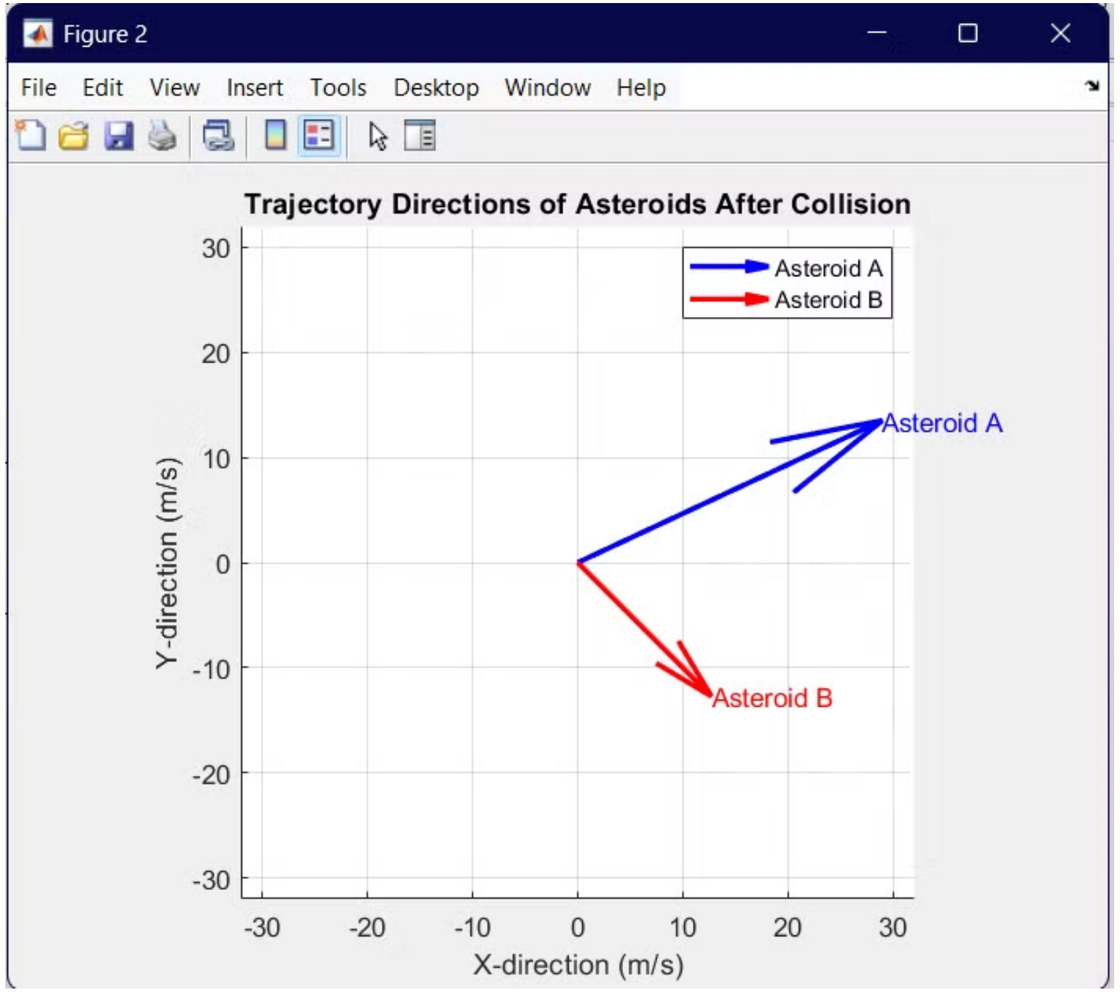
Collision of Asteroids

```
>> % MATLAB Script for Plotting Asteroid Trajectories

% Input: Final velocities and angles (in degrees)
vA = input('Enter the final velocity of Asteroid A (m/s): ');
vB = input('Enter the final velocity of Asteroid B (m/s): ');
thetaA = input('Enter the deflection angle of Asteroid A (degrees): ');
thetaB = input('Enter the angle of Asteroid B after collision (degrees): ');

% Convert angles to radians
thetaA_rad = deg2rad(thetaA);
thetaB_rad = deg2rad(thetaB);

% Compute the components of velocity for both asteroids
vA_x = vA * cos(thetaA_rad);
vA_y = vA * sin(thetaA_rad);

vB_x = vB * cos(thetaB_rad);
vB_y = vB * sin(thetaB_rad);

% Plot the trajectories
figure;
hold on;
grid on;
```

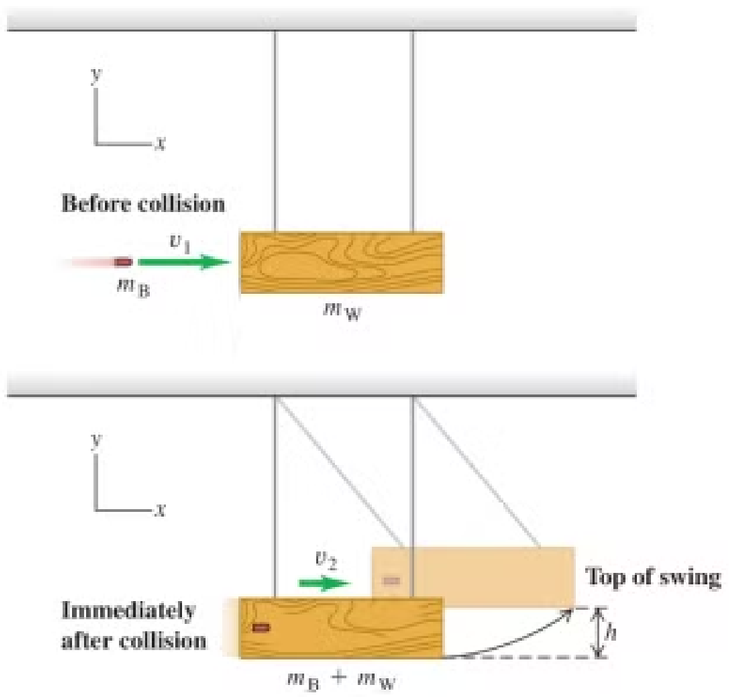
```
% Plot the trajectory of Asteroid A
quiver(0, 0, vA_x, vA_y, 0, 'b', 'LineWidth', 2, 'MaxHeadSize', 2);
text(vA_x, vA_y, 'Asteroid A', 'FontSize', 10, 'Color', 'blue');

% Plot the trajectory of Asteroid B
quiver(0, 0, vB_x, vB_y, 0, 'r', 'LineWidth', 2, 'MaxHeadSize', 2);
text(vB_x, vB_y, 'Asteroid B', 'FontSize', 10, 'Color', 'red');

% Add labels and title
xlabel('X-direction (m/s)');
ylabel('Y-direction (m/s)');
title('Trajectory Directions of Asteroids After Collision');
legend('Asteroid A', 'Asteroid B');

% Set axis limits for better visibility
axis equal;
xlim([-max(vA, vB) max(vA, vB)]);
ylim([-max(vA, vB) max(vA, vB)]);

hold off;
Enter the final velocity of Asteroid A (m/s): 32
Enter the final velocity of Asteroid B (m/s): 18
Enter the deflection angle of Asteroid A (degrees): 25
Enter the angle of Asteroid B after collision (degrees): 45
```

Graphs and Simulations

After running the code in matlab :



Task 3: Ballistic Pendulum

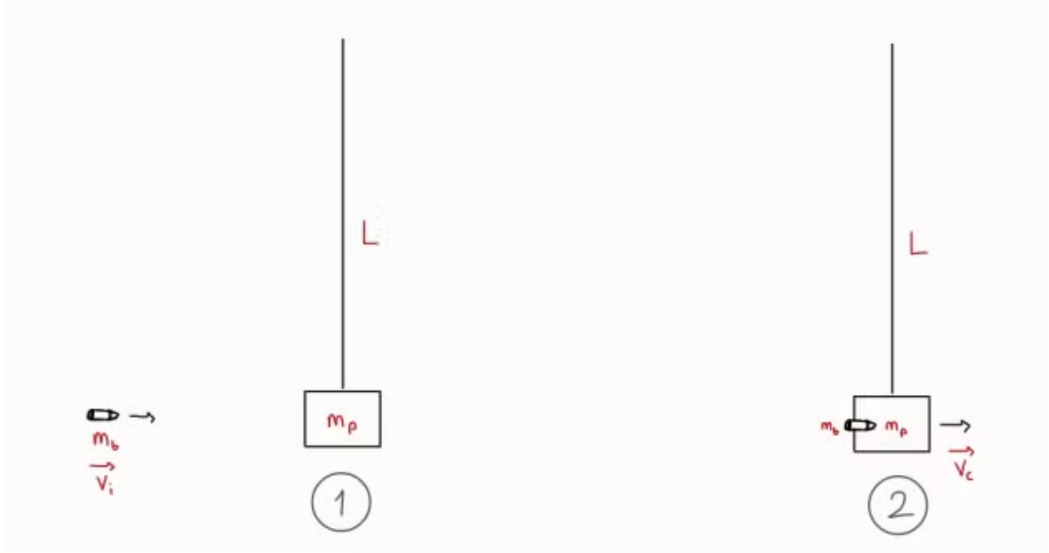


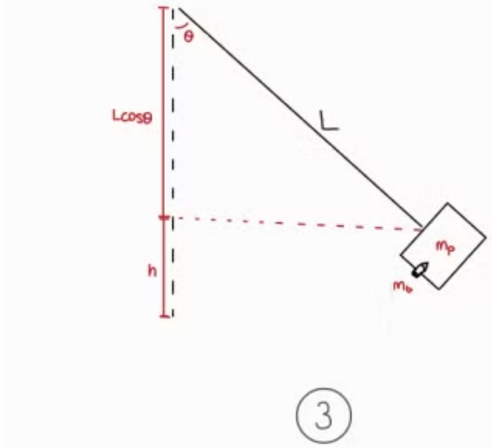
A bullet is fired with a speed into a pendulum that is a distance ahead

- calculating the height after bullet gets into the pendulum
- calculate the initial kinetic energy and the combined kinetic energy after the collision
- simulate pendulum's path

Physics Formulas

Ballistic Pendulum





$$P_i = P_f \qquad \sum E_c = \sum E_F$$

$$m_b \cdot \vec{V_i} = (m_b + m_p) \cdot \vec{V_c}$$

$$\vec{V_c} = \frac{m_b}{m_b + m_p} \cdot \vec{V_i}$$

$$\frac{1}{2}(m_b + m_p) \cdot V_c^2 = (m_b + m_p) \cdot g \cdot h$$

$$h = \frac{1}{2g} \cdot V_c^2$$

$$KE_i = \frac{1}{2} m_b \cdot V_i^2$$

$$KE_c = \frac{1}{2}(m_b + m_c) \cdot V_c^2$$

$$h = L - L\cos\theta$$

$$h = L(1 - \cos\theta)$$

$$\frac{h}{L} = 1 - \cos\theta$$

$$\cos\theta = 1 - \frac{h}{L}$$

$$\theta = \cos^{-1}\left(1 - \frac{h}{L}\right)$$

Algorithm and flowchart

1-Data Entry:

- Get the mass of the bullet in grams from the user.
- Get the speed of the bullet in meters per second from the user.
- Get the mass of the pendulum in kilograms from the user.
- Get the length of the pendulum in centimeters from the user.

2-Converting Units:

- Convert the mass of the bullet from grams to kilograms.
- Change the length of the pendulum from centimeters to meters.

3-Calculate Post-Collision Speed:

- Calculate post-collision velocity using the principle of momentum conservation:

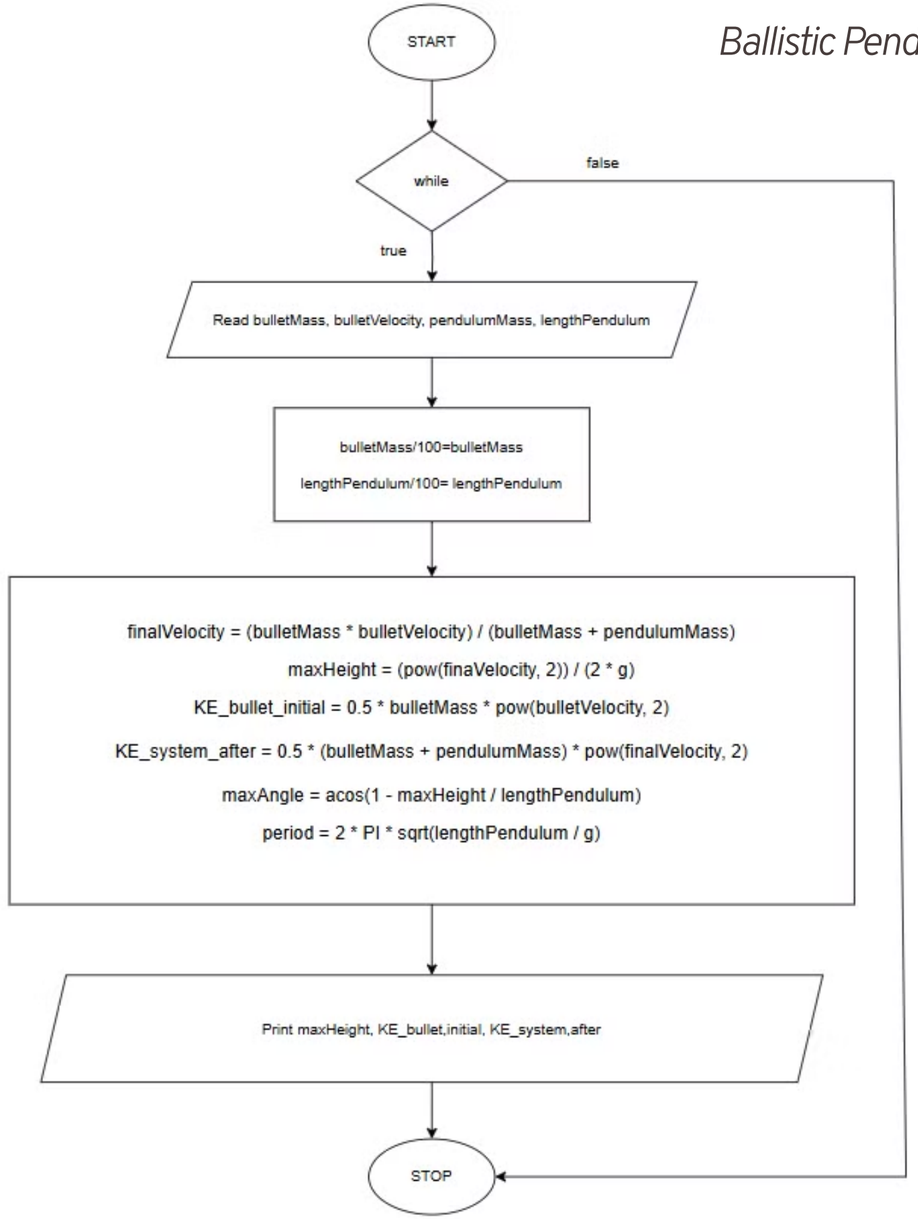
finalVelocity = (bulletMass * bulletVelocity) / (bulletMass + pendulumMass)

4-Calculate Maximum Height:

- Calculate the maximum height after the collision by converting kinetic energy into potential energy:

maxHeight = ((finaVelocity)^2) / (2 * g)

Ballistic Pendulum



Algorithm and flowchart

5-Calculate the Initial Kinetic Energy of the Projectile:

- Calculate the initial kinetic energy of the projectile:
  KE_bullet_initial = 0.5 * bulletMass * (bulletVelocity)^2)

6-Kinetic Energy After Collision:

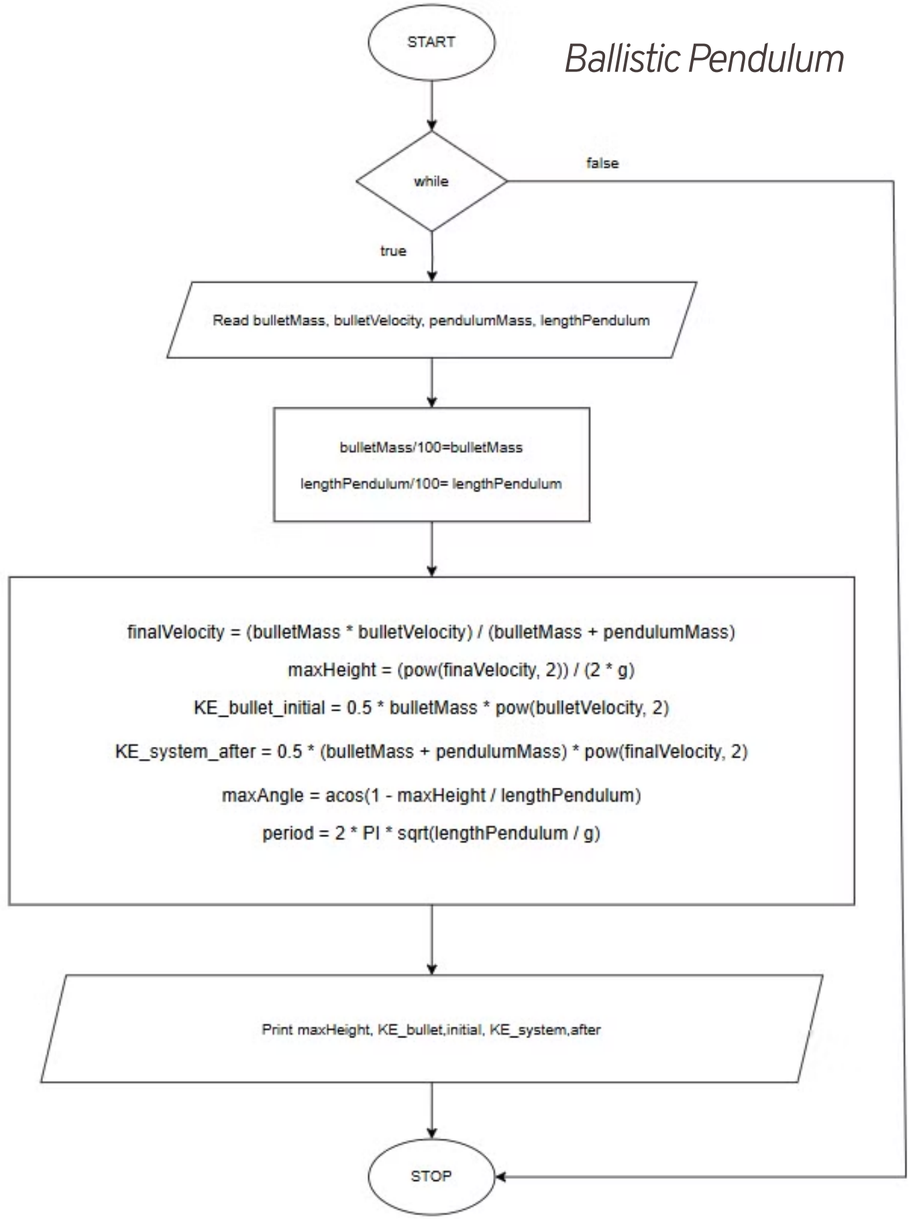
- Calculate the total kinetic energy of the projectile and the pendulum after the collision:
  KE_system_after = 0.5 * (bulletMass + pendulumMass) * (finalVelocity)^2)

7-Calculate Maximum Angle and Release Time:

- Calculate the maximum angle:
  Θmax = cos^-1(1-maxHeight/ lengthPendulum)
- Calculate the pendulum swing time:
  period = 2 * PI * √(lengthPendulum / g)

8-Print Results on Screen:

- Print the maximum height.
- Print the initial kinetic energy of the projectile.
- Print kinetic energy after impact.



C++ Codes

functions

```cpp
void unit_conversion1(double& value) {
    //g to kg
    value = value / 1000;
}

void unit_conversion2(double& value) {
    //cm to m
    value = value / 100;
}

double max_angle(double maxHeight, double lengthPendulum) {
    double ratio = 1 - (maxHeight / lengthPendulum);

    // checks if the value is valid
    if (ratio < -1.0 || ratio > 1.0) {
        cout << "\a";  // alarm sounds if the value is invalid
        cout << "Error: Invalid ratio (" << ratio << "). The ratio must be between -1 and 1." << endl;
        return -1; // -1 is returned on error
    }
    return acos(ratio);
}
```

C++ Codes

main

```cpp
void pendulum(){

    cout << "============================================================" << endl;
    cout << "        Welcome to the Ballistic Pendulum Simulator         " << endl;
    cout << "============================================================" << endl;

    cout << "\nthe aim of this application is to calculate the change over heigh of a pendulum when it gets hit by a bullet\n";
    cout << "initial kinetic energy of bullet and kinetic energy of combined body at collision moment will be printed as output as well\n";
    cout << "now to get started to calculations\n";


    string ans = "yes";
    while(ans == "yes"){
    double bulletMass, bulletVelocity, pendulumMass, lengthPendulum, finalVelocity, maxHeight, KE_bullet_initial, KE_system_after, maxAngle, period;

    cout << endl;

    // we get the required values from the user
    bulletMass = getPositiveInput("Please enter mass of bullet (in g): ");
    bulletVelocity = getPositiveInput("Please enter velocity of bullet (in m/s): ");
    pendulumMass = getPositiveInput("Please enter mass of pendulum (in g): ");
    lengthPendulum = getPositiveInput("Please enter length of pendulum (in cm): ");

    // we make unit conversions
    unit_conversion1(bulletMass); // kg
    unit_conversion1(pendulumMass); // kg
    unit_conversion2(lengthPendulum); // m

    // post collision speed and height calculations
    KE_bullet_initial = 0.5 * bulletMass * pow(bulletVelocity, 2);
    finalVelocity = (bulletMass * bulletVelocity) / (bulletMass + pendulumMass);
    maxHeight = (pow(finalVelocity, 2)) / (2 * g);
    KE_system_after = 0.5 * (bulletMass + pendulumMass) * pow(finalVelocity, 2);
    period = 2 * PI * sqrt(lengthPendulum / g);

    // we calculate the maximum angle
    maxAngle = max_angle(maxHeight, lengthPendulum);
```

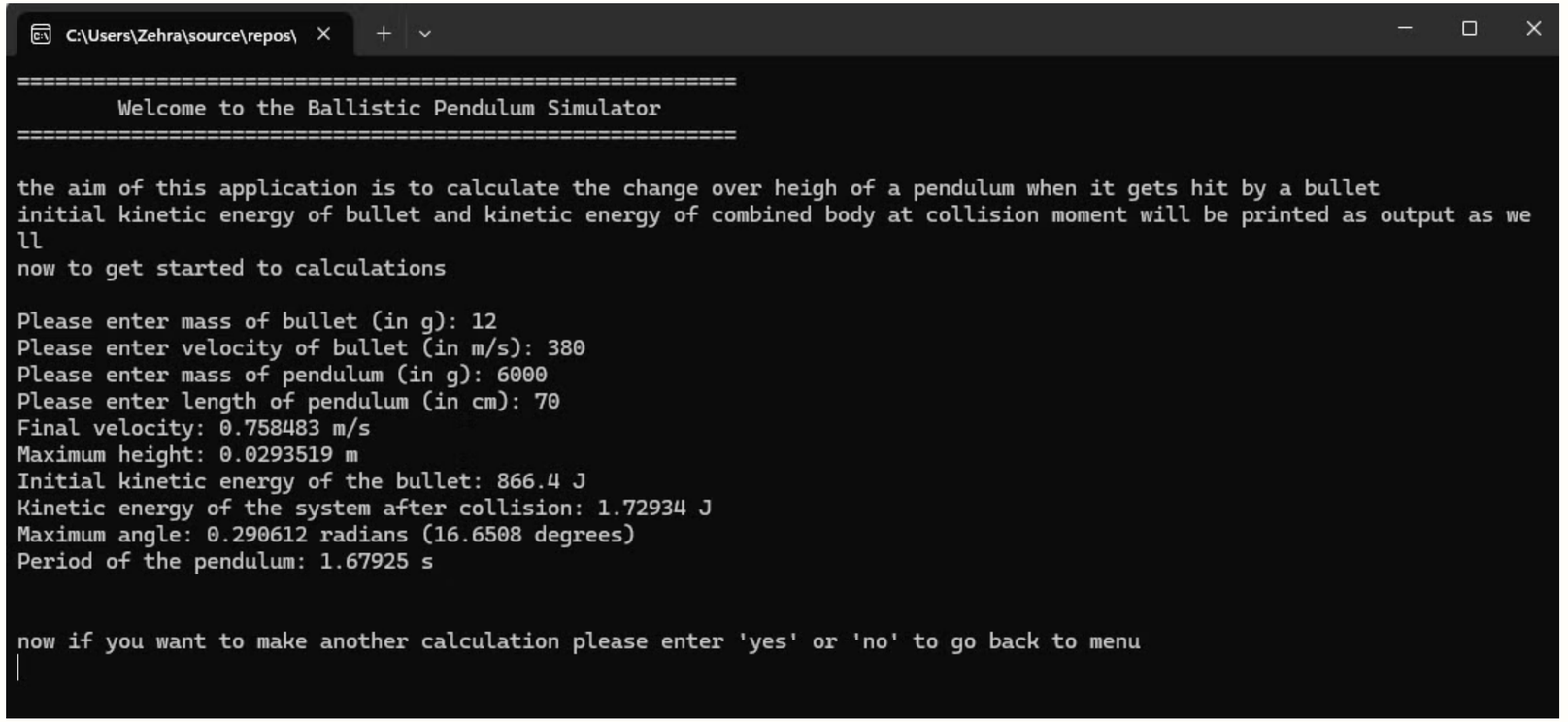
C++ Codes

main

```
    // error status check
    if (maxAngle == -1) {
        cout << "\aCalculation failed due to invalid ratio." << endl;
        continue;
    // Program terminates in case of error
    }

    // we print the results on the screen
    cout << "Final velocity: " << finalVelocity << " m/s" << endl;
    cout << "Maximum height: " << maxHeight << " m" << endl;
    cout << "Initial kinetic energy of the bullet: " << KE_bullet_initial << " J" << endl;
    cout << "Kinetic energy of the system after collision: " << KE_system_after << " J" << endl;
    cout << "Maximum angle: " << maxAngle << " radians (" << maxAngle * (180.0 / PI) << " degrees)" << endl;
    cout << "Period of the pendulum: " << period << " s" << endl; cout << endl;


    cout << "\nnow if you want to make another calculation please enter 'yes' or 'no' to go back to menu\n";
    cin >> ans;
    for(auto& c: ans) c = tolower (c);

    while(ans != "yes" && ans != "no"){
        cout << "please enter 'yes' or 'no'\n";
        std::cin >> ans;
        for(auto& c: ans) c = tolower (c);
    }

    system("cls");

    }

    menu();
}
```

Console

Ballistic Pendulum



Graphs and Simulations

Ballistic Pendulum

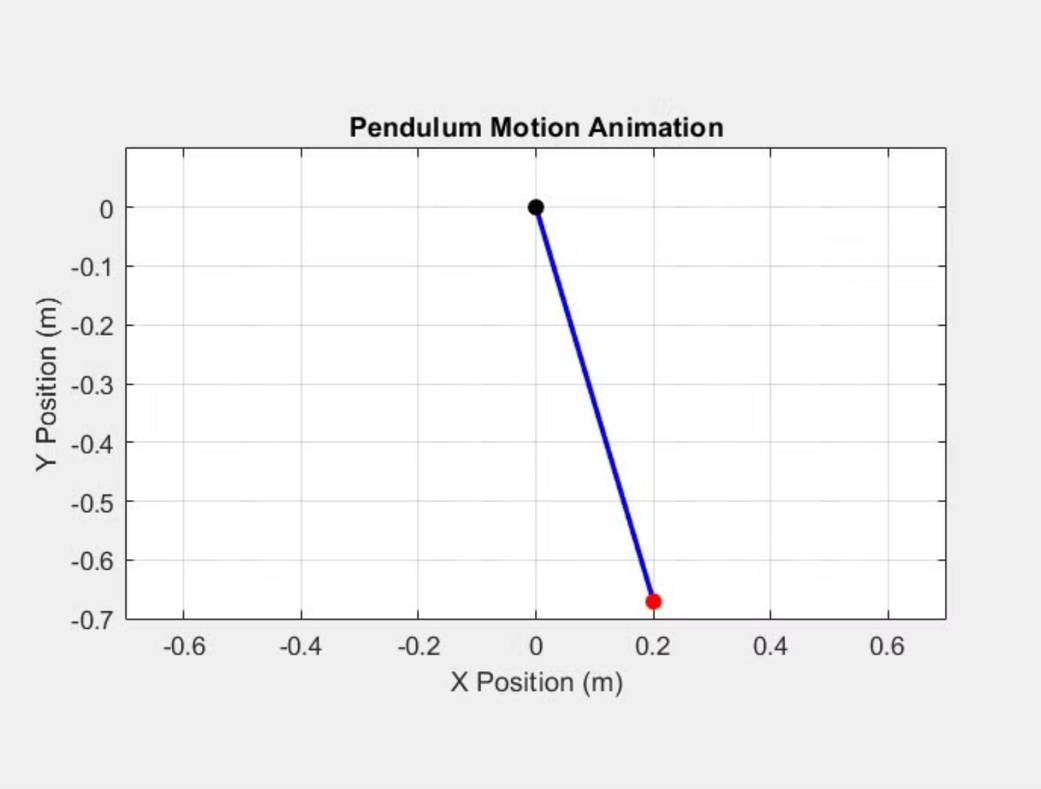
```
% Simulation Parameters
bulletMass_g = input('Please enter mass of bullet (in g): '); % g
bulletVelocity = input('Please enter velocity of bullet (in m/s): '); % m/s
pendulumMass_g = input('Please enter mass of pendulum (in g): '); % g
lengthPendulum_cm = input('Please enter length of pendulum (in cm): '); % cm

% Unit Conversions
bulletMass = bulletMass_g / 1000; % kg
pendulumMass = pendulumMass_g / 1000; % kg
lengthPendulum = lengthPendulum_cm / 100; % m

% Constants
g = 9.8; % m/s^2
PI = 3.14159;

% Post-Collision Velocity and Height Calculations
KE_bullet_initial = 0.5 * bulletMass * bulletVelocity^2;
finalVelocity = (bulletMass * bulletVelocity) / (bulletMass + pendulumMass);
maxHeight = (finalVelocity^2) / (2 * g);
KE_system_after = 0.5 * (bulletMass + pendulumMass) * finalVelocity^2;
period = 2 * PI * sqrt(lengthPendulum / g);
```

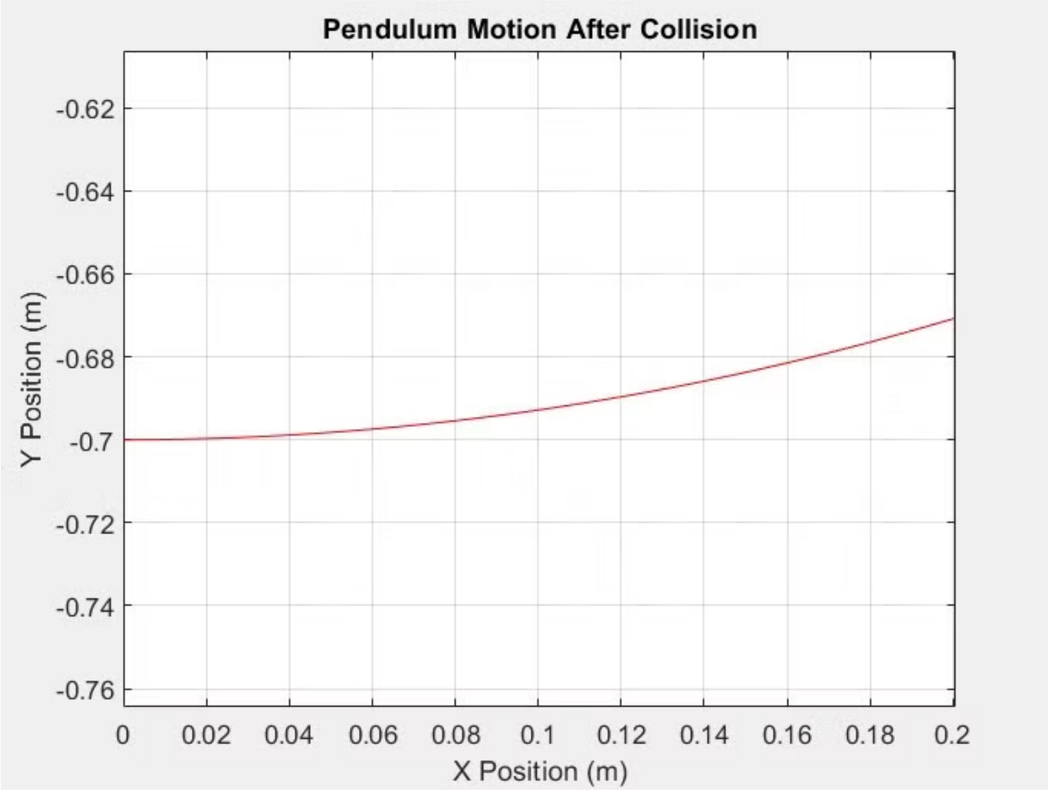
```
% Maximum Angle Calculation
ratio = 1 - (maxHeight / lengthPendulum);
if ratio < -1.0 || ratio > 1.0
    error('Invalid ratio. The ratio must be between -1 and 1.');
end
maxAngle = acos(ratio);

% Time Interval for Simulation (rightward movement only)
t_max = period / 4; % time to reach maximum height (quarter period)
t = linspace(0, t_max, 500); % time interval
theta = maxAngle * sin(sqrt(g / lengthPendulum) * t); % angle calculation (rightward swing only)

% Pendulum Motion Simulation
x = lengthPendulum * sin(theta);
y = -lengthPendulum * cos(theta);

% Plotting
figure;
plot(x, y, 'r');
xlabel('X Position (m)');
ylabel('Y Position (m)');
title('Pendulum Motion After Collision');
axis equal;
grid on;
```

Graphs and Simulations

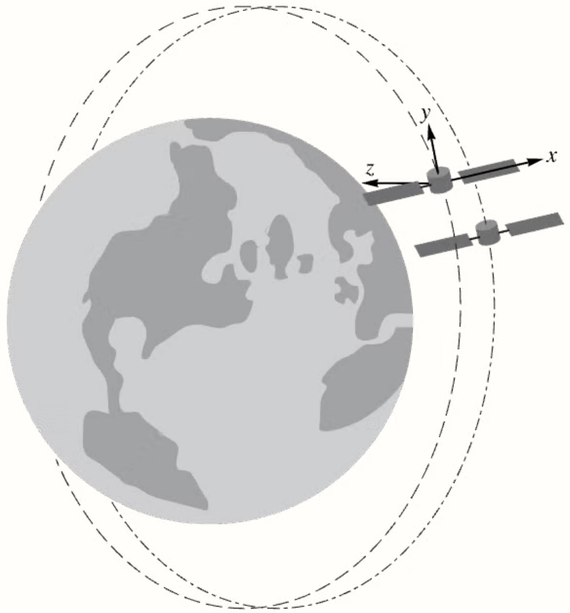

```
% Pendulum Motion Animation
figure;
% Speed-up factor
speed_up_factor = 5; % You can adjust the desired speed-up factor here

% Time scale calculation
real_time_per_second = speed_up_factor; % each second in animation represents this many real-world seconds
fprintf('Time scale: 1 second in animation = %.2f real-world seconds\n', real_time_per_second);

for i = 1:length(t)
    plot([0 x(i)], [0 y(i)], 'b-', 'LineWidth', 2); % pendulum line
    hold on;
    plot(x(i), y(i), 'ro', 'MarkerFaceColor', 'r'); % pendulum mass
    plot(0, 0, 'ko', 'MarkerFaceColor', 'k'); % pivot point
    hold off;
    axis equal;
    axis([-lengthPendulum lengthPendulum -lengthPendulum 0.1]);
    xlabel('X Position (m)');
    ylabel('Y Position (m)');
    title('Pendulum Motion Animation');
    grid on;
    drawnow;
    pause(t_max / length(t) / speed_up_factor); % reduce pause time to speed up the animation
end
```

Graphs and Simulations





Task 4: Satellite Motion Simulation

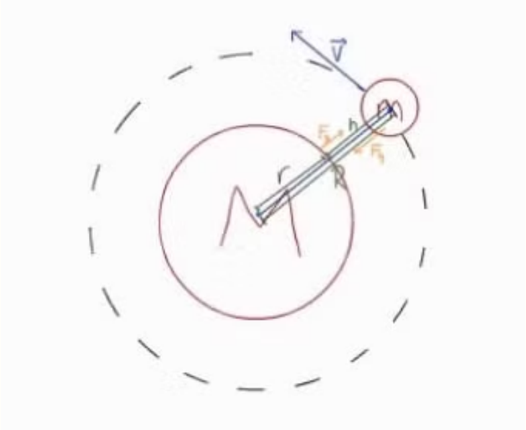


Satellites are tasked to orbit their planets

- Calculate satellite's orbital velocity and period

Physics Formulas

Satellite Motion Simulation



$$R = r + h$$

$$F_g = F_c$$

$$F_g = \frac{G \cdot M \cdot m}{R^2} \qquad F_c = m \cdot a_c = \frac{mV^2}{R}$$

$$\frac{G \cdot M \cdot m}{R^2} = \frac{mV^2}{R}$$

$$\frac{GM}{R} = V^2$$

$$V = \sqrt{\frac{GM}{R}}$$

$$\boxed{V = \sqrt{\frac{GM}{r+h}}}$$

$$V = \omega \cdot R$$

$$\sqrt{\frac{GM}{R}} = \frac{2\pi}{T} \cdot R$$

$$T = 2\pi R \sqrt{\frac{R}{GM}}$$

$$T = \sqrt{\frac{4\pi^2 R^3}{GM}}$$

$$\boxed{T = \sqrt{\frac{4\pi^2 (r+h)^3}{GM}}}$$

Algorithm and flowchart

1-Data Entry
- Define G as 6.6743 * 10^-11
- Define pi as 3.14159

2-User Input
- Get the mass of orbit in kilograms from the user
- Get the radius of orbit
- Get the length between satellite and surface of the orbit

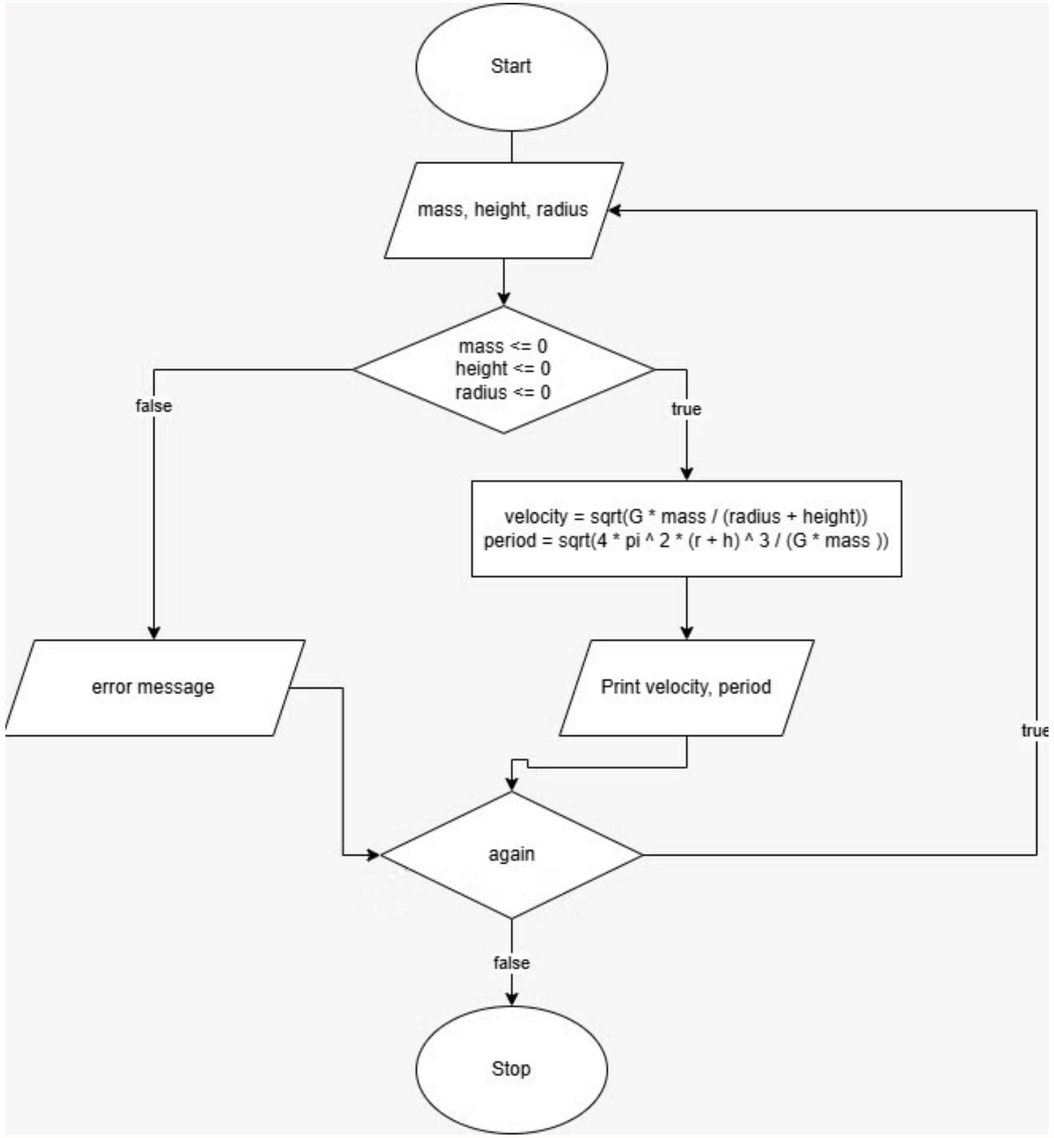
3-Calculate Revolution Speed
- Calculate velocity with using sqrt(mass * G / (altitude + radius))

4-Calculate Revolution Period
- Calculate revolution period with using sqrt(4 * pi^2 * (altitude + radius)^3 / (G * mass))

5-Print the Results as Output
- Print revolution speed
- Print revolution period

6-User Input
- Ask the user if they want to make another calculation

7-Check the Condition
- If answer is yes go to step 2
- If answer is no stop the program



C++ Codes

functions

```
430     //gives the velocity of satellite
431     float velocitycalculation(float radius, float altitude, float mass){
432
433         return sqrt((mass * G) / (altitude + radius));
434     }
435
436     //gives the period of revolution of satellite
437     float periodcalculation(float radius, float altitude, float mass){
438
439         return sqrt((4 * PI * PI * pow((altitude + radius), 3)) / (G * mass));
440     }
```

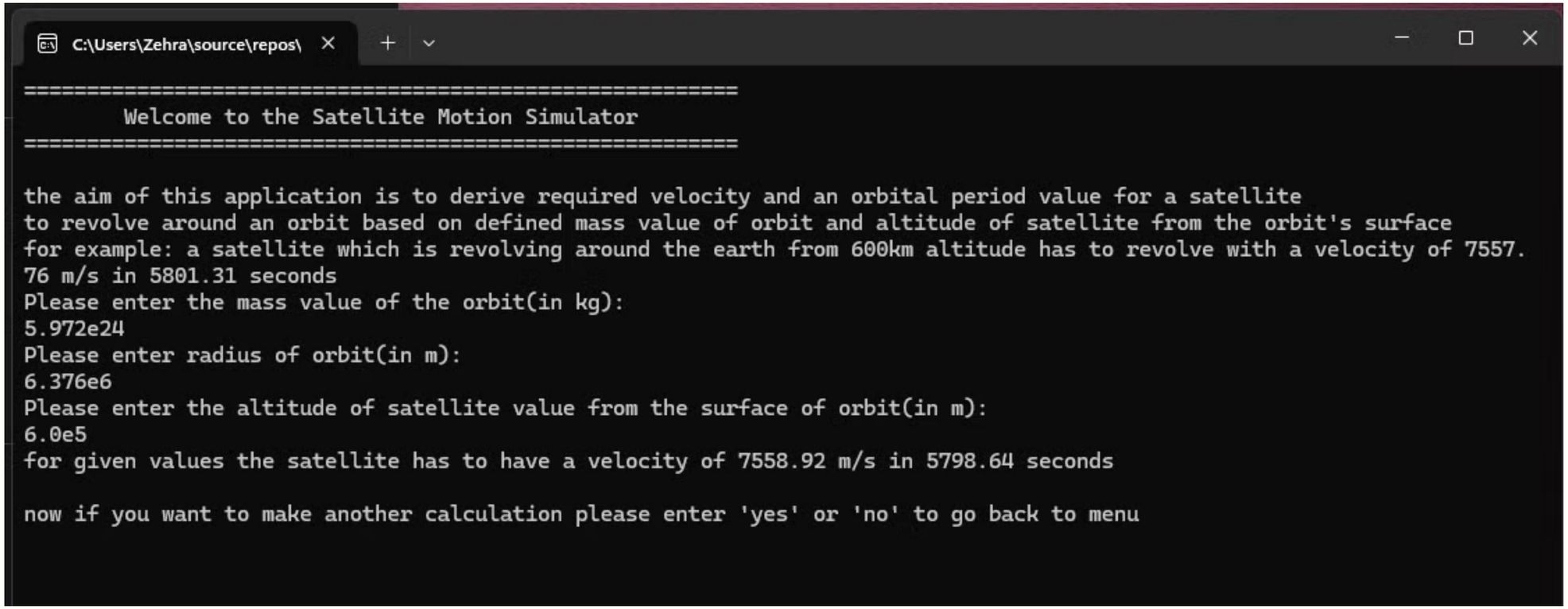
C++ Codes

main

```cpp
void satellite(){

    cout << "=========================================================" << endl;
    cout << "        Welcome to the Satellite Motion Simulator        " << endl;
    cout << "=========================================================" << endl;

    const int altitude_from_earth = 600000;
    const float mass_of_world = 5.972 * pow(10, 24);
    const int radius_of_world = 6378137;
    float altitude;
    float radius;
    float mass;
    string ans;

    float velocity;
    float period;

    //an introduction message to explain the aim of application and how it works
    cout << "\nthe aim of this application is to derive required velocity and an orbital period value for a satellite\n";
    cout << "to revolve around an orbit based on defined mass value of orbit and altitude of satellite from the orbit's surface\n";
    cout << "for example: a satellite which is revolving around the earth from 600km altitude has to revolve with a velocity of ";
    cout << velocitycalculation(radius_of_world, altitude_from_earth, mass_of_world) << " m/s in " << periodcalculation(radius_of_world, altitude_from_earth, mass_of_world) << " seconds\n";


    do{
        //ask user for required inputs

        mass = getPositiveInput("Please enter the mass value of the orbit(in kg):\n");
        radius = getPositiveInput("Please enter radius of orbit(in m):\n");
        altitude = getPositiveInput("Please enter the altitude of satellite value from the surface of orbit(in m):\n");

        //output
        cout << "for given values the satellite has to have a velocity of " << velocitycalculation(radius, altitude, mass) << " m/s in " << periodcalculation(radius, altitude, mass) << " seconds\n";

        //ask user if they want to make another calculation
        cout << "\nnow if you want to make another calculation please enter 'yes' or 'no' to go back to menu\n";
        cin >> ans;
        for(auto& c: ans) c = tolower (c);
        while(ans != "yes" && ans != "no"){
            cout << "please enter 'yes' or 'no'\n";
            std::cin >> ans;
            for(auto& c: ans) c = tolower (c);
        }

        system("cls");

    }while (ans == "yes");

    menu();

}
```

Console

Satellite Motion Simulation



Graphs and Simulations


```
% Constants
PI = 3.14159;
G = 6.6743e-11;

% Function to validate and get positive inputs
function value = getPositiveInput(prompt)
    value = input(prompt);
    while value <= 0
        disp('Oops! The value must be positive. Please try again.');
        value = input(prompt);
    end
end

% Function to calculate the velocity of the satellite
function velocity = velocityCalculation(radius, altitude, mass)
    G = 6.6743e-11; % Gravitational constant
    velocity = sqrt((mass * G) / (altitude + radius));
end

% Function to calculate the orbital period of the satellite
function period = periodCalculation(radius, altitude, mass)
    G = 6.6743e-11; % Gravitational constant
    PI = 3.14159;   % Define PI
    period = sqrt((4 * PI^2 * (altitude + radius)^3) / (G * mass));
end
```

```
% Main Script
clc;
disp('============================================================');
disp('        Welcome to the Satellite Motion Simulator         ');
disp('============================================================');

% Introduction message
disp('The aim of this application is to derive required velocity');
disp('and orbital period values for a satellite to revolve');
disp('around an orbit based on the defined mass and altitude.');

while true
    % Input
    mass = getPositiveInput('Please enter the mass value of the orbit (in kg): ');
    radius = getPositiveInput('Please enter the radius of the orbit (in m): ');
    altitude = getPositiveInput('Please enter the altitude of the satellite from the surface of the orbit (in m): ');

    % Calculations
    velocity = velocityCalculation(radius, altitude, mass);
    period = periodCalculation(radius, altitude, mass);

    % Output
    fprintf('For the given values:\n');
    fprintf(' - Velocity: %.2f m/s\n', velocity);
    fprintf(' - Orbital Period: %.2f seconds\n', period);
```

Graphs and Simulations

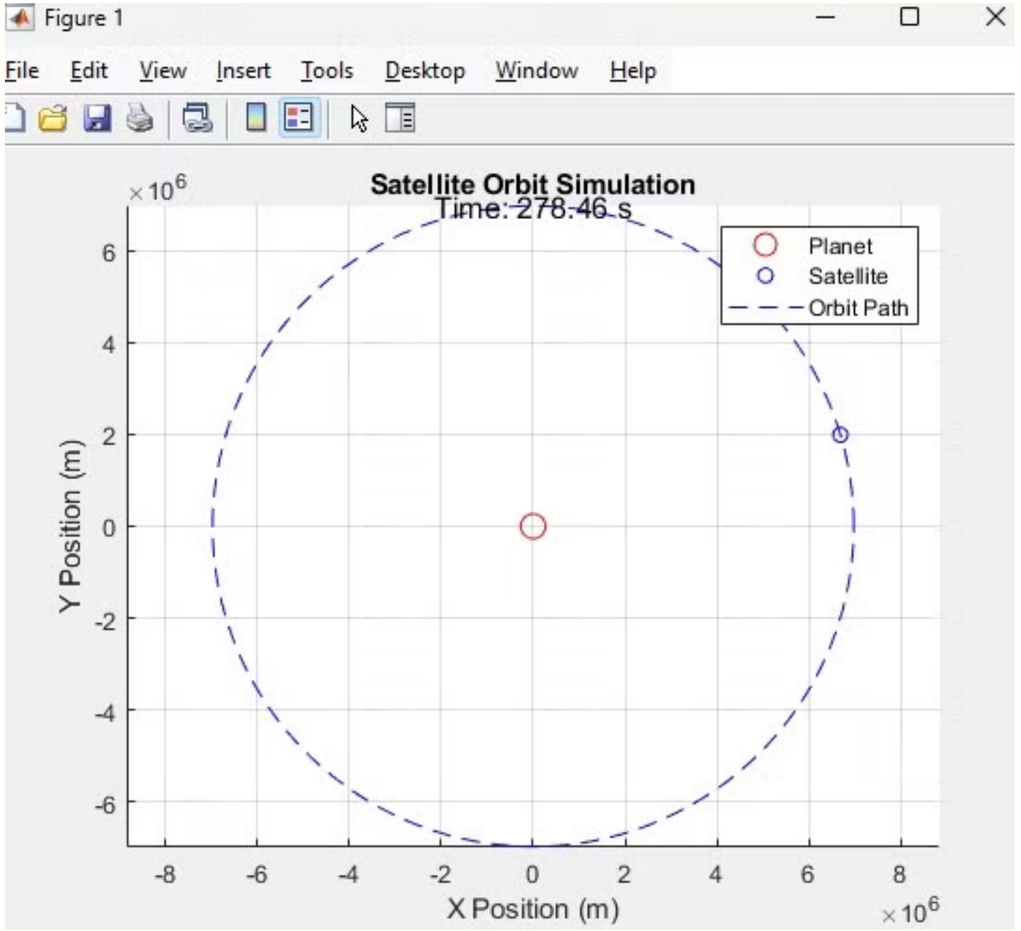
Satellite Motion Simulation

```
% Visualization
orbitRadius = radius + altitude; % Total radius of the orbit
numPoints = 500; % Number of points for smooth simulation
theta = linspace(0, 2 * pi, numPoints);

xOrbit = orbitRadius * cos(theta);
yOrbit = orbitRadius * sin(theta);

% Check if the figure already exists, if not, create it
figureHandle = findobj('Type', 'figure');
if isempty(figureHandle)
    figure; % Create a new figure if not already created
else
    figure(figureHandle); % Use the existing figure
end
hold on;
plot(0, 0, 'ro', 'MarkerSize', 10, 'DisplayName', 'Planet'); % Planet at the center
satellite = plot(xOrbit(1), yOrbit(1), 'bo', 'MarkerSize', 6, 'DisplayName', 'Satellite'); % Satellite
plot(xOrbit, yOrbit, 'b--', 'DisplayName', 'Orbit Path'); % Circular orbit path
axis equal;
legend;
title('Satellite Orbit Simulation');
xlabel('X Position (m)');
ylabel('Y Position (m)');
grid on;
```

```
% Time display setup
timeDisplay = text(0, orbitRadius + 10, 'Time: 0 s', 'FontSize', 12, 'Color', 'black', 'HorizontalAlignment', 'center');

% Animation Time Scale Factor
timeScale = 100; % Adjust this value to control the speed of the animation (higher = faster)

% Animation Loop
for i = 1:numPoints
    % Set the satellite position
    satellite.XData = xOrbit(i);
    satellite.YData = yOrbit(i);

    % Calculate the "real" time at this position in the orbit
    realTime = (i / numPoints) * period; % Real time (seconds)

    % Update the time display
    timeDisplay.String = sprintf('Time: %.2f s', realTime);

    % Pause for the scaled animation period
    pause((period / numPoints) / timeScale); % Control animation speed based on orbital period
end
```

Graphs and Simulations


```
    % Ask user if they want another calculation
    userResponse = input('Do you want to make another calculation? Enter "yes" or "no": ', 's');
    userResponse = lower(userResponse);
    while ~strcmp(userResponse, 'yes') && ~strcmp(userResponse, 'no')
        disp('Please enter "yes" or "no".');
        userResponse = input('Do you want to make another calculation? Enter "yes" or "no": ', 's');
    end
    if strcmp(userResponse, 'no')
        break;
    end
end

disp('Thank you for using the Satellite Motion Simulator!');
```



Thanks for listening.

Prepared by:

Yasemin Yavuz
Fatma Zehra Keşkek
Rohat Nur Uzan
Enes Erşan
Muhammed Emir Kaş

SİVAS
BİLİM VE TEKNOLOJİ
ÜNİVERSİTESİ

Pitch

Want to make a presentation like this one?

Start with a fully customizable template, create a beautiful deck in minutes, then easily share it with anyone.

Create a presentation (It's free)