Projectile Motion and Drag Force

New Mexico
Supercomputing Challenge
Final Report
April 3rd, 2019

Team Number 21
La Cueva High School

Team Members
David Feng
Blake Watson
Luke Xue
Brad Zhang

Teacher
Mrs. Terri Jones

Executive Summary

Problem Investigated

The goal was to create an accurate physics model that would be able to predict landing sites of projectiles based on a number of factors, including wind speed, surface area, launch velocity and angle, and wind direction and speed. This has a number of application, including space launch cases, sports application, and military applications. Another common problem is to find the launch angle and velocity based upon the crash site, which our program is able to do.

Solution Method

Most of our equations focus upon the drag force of an object. The drag force is the main factor differentiating our program from extremely simplistic physics models. It can be expressed as F_D and is always a vector working in the opposite direction of the velocity vector. With these factors in mind, the Drag Force Magnitude equation can be expressed as: $F_D = \frac{1}{2} C_D \rho A v^2$, where C_D is the drag coefficient, ρ is the air density, A is the cross sectional area of the projectile object, and v is the object's speed. All of these are in standard units for their respective values. Based upon this equation and knowing that Drag force works directly against the velocity of an object, the Net Force equation can be shown as $\Sigma F = mg - \frac{1}{2} C_D \rho A v^2$ where both g and F are vectors in the same direction as the velocity vector of the object. Using this and the vector object library found within C++, our program accurately calculates the projected landing site of the object when given a number of variables

Test Verification

Even though our methods were based upon widely-used physics laws and equations, we attempted to verify test data through a number of ways. Manual checking of numbers was used to ensure that our program could utilize any known equations properly, and a number of other online tools and software that accomplishes similar goals was used to compare results. While we did not modify results based upon other numbers found by other calculators and software, we did include them in our results as a baseline to be compared to our own results. While other software attempts to accomplish the same goal, the discrepancies in outputs can often be explained through a combination of differing techniques and perhaps a more simplistic equation and method that is not suited to take into account wind resistance and drag force.

Test Cases: Single-Launch

Test 1 (Control)

```
                Mass:         40 kilograms
    Drag Coefficient:       0.47
         Air Density:      1.225 kg/m^3
Cross-sectional Area:       0.15 m^2

        Launch Speed:         20 m/s
        Launch Angle:         35 degrees above horizontal
    Launch Direction:          0 degrees
          Wind Speed:          0 m/s
      Wind Direction:          0 degrees

               Range:    37.2711 meters (website data shows 37.45)
   Landing Direction:          0 degrees
            Air Time:      2.325 seconds
```

Test 2 (lower mass than Test 1)

```
                Mass:          4 kilograms
    Drag Coefficient:       0.47
         Air Density:      1.225 kg/m^3
Cross-sectional Area:       0.15 m^2

        Launch Speed:         20 m/s
        Launch Angle:         35 degrees above horizontal
```

```
    Launch Direction:          0 degrees
          Wind Speed:          0 m/s
      Wind Direction:          0 degrees

               Range:    29.8808 meters (website data shows 31.18)
   Landing Direction:          0 degrees
            Air Time:      2.184 seconds
```

Test 3 (greater cross-sectional area than Test 1)

```
                Mass:         40 kilograms
    Drag Coefficient:       0.47
         Air Density:      1.225 kg/m^3
Cross-sectional Area:         10 m^2

        Launch Speed:         20 m/s
        Launch Angle:         35 degrees above horizontal
    Launch Direction:          0 degrees
          Wind Speed:          0 m/s
      Wind Direction:          0 degrees

               Range:    14.7376 meters (website data shows 16.637)
   Landing Direction:          0 degrees
            Air Time:       1.72 seconds
```

Test 4 (greater launch speed than Test 1)

```
                Mass:         40 kilograms
    Drag Coefficient:       0.47
         Air Density:      1.225 kg/m^3
Cross-sectional Area:       0.15 m^2

        Launch Speed:         50 m/s
        Launch Angle:         35 degrees above horizontal
    Launch Direction:          0 degrees
          Wind Speed:          0 m/s
      Wind Direction:          0 degrees

               Range:    203.011 meters (website data shows 209.07)
   Landing Direction:          0 degrees
            Air Time:      5.594 seconds
```

Test 5 (lower launch angle than Test 1)

```
                Mass:         40 kilograms
    Drag Coefficient:       0.47
```

```
         Air Density:      1.225 kg/m^3
Cross-sectional Area:       0.15 m^2

        Launch Speed:         20 m/s
        Launch Angle:         25 degrees above horizontal
    Launch Direction:          0 degrees
          Wind Speed:          0 m/s
      Wind Direction:          0 degrees

               Range:    30.5762 meters (website data shows 30.69)
   Landing Direction:          0 degrees
            Air Time:      1.716 seconds
```

Test 6 (greater launch angle than Test 1)

```
                Mass:         40 kilograms
    Drag Coefficient:       0.47
         Air Density:      1.225 kg/m^3
Cross-sectional Area:       0.15 m^2

        Launch Speed:         20 m/s
        Launch Angle:         45 degrees above horizontal
    Launch Direction:          0 degrees
          Wind Speed:          0 m/s
      Wind Direction:          0 degrees

               Range:    39.4775 meters (website data shows 39.73)
   Landing Direction:          0 degrees
            Air Time:      2.861 seconds
```

Test 7 (greater launch angle than Test 6)

```
                Mass:         40 kilograms
    Drag Coefficient:       0.47
         Air Density:      1.225 kg/m^3
Cross-sectional Area:       0.15 m^2

        Launch Speed:         20 m/s
        Launch Angle:         55 degrees above horizontal
    Launch Direction:          0 degrees
          Wind Speed:          0 m/s
      Wind Direction:          0 degrees

               Range:    37.0302 meters (website data shows 37.31)
   Landing Direction:          0 degrees
```

```
           Air Time:      3.312 seconds
```

<u>Test 8 (Test 1 without air resistance)</u>

```
               Mass:         40 kilograms
   Drag Coefficient:       0.47
        Air Density:      1.225 kg/m^3
Cross-sectional Area:      0.15 m^2

       Launch Speed:         20 m/s
       Launch Angle:         35 degrees above horizontal
   Launch Direction:          0 degrees
         Wind Speed:          0 m/s
     Wind Direction:          0 degrees

              Range:    38.3855 meters (website says 38.36)
  Landing Direction:          0 degrees
           Air Time:      2.343 seconds
```

<u>Test 9 (Test 2 without air resistance) (lower mass than Test 8)</u>

```
               Mass:          4 kilograms
   Drag Coefficient:       0.47
        Air Density:      1.225 kg/m^3
Cross-sectional Area:      0.15 m^2

       Launch Speed:         20 m/s
       Launch Angle:         35 degrees above horizontal
   Launch Direction:          0 degrees
         Wind Speed:          0 m/s
     Wind Direction:          0 degrees

              Range:    38.3855 meters (website says 38.36)
  Landing Direction:          0 degrees
           Air Time:      2.343 seconds
```

Prediction Cases

```
               Mass:          5 kilograms
   Drag Coefficient:       0.47
        Air Density:      1.225 kg/m^3
Cross-sectional Area:      0.25 m^2
```

<u>Case 1: No wind (Control)</u>

```
       Launch Speed:         20 m/s
       Launch Angle:         40 degrees above horizontal
```

```
 Launch Direction:         0 degrees
       Wind Speed:         0 m/s
   Wind Direction:         0 degrees

            Range:   28.6682 meters
Landing Direction:         0 degrees
         Air Time:     2.381 seconds
```

Case 2: 10 m/s Wind at 45 degrees from launch direction

```
     Launch Speed:        20 m/s
     Launch Angle:        40 degrees above horizontal
 Launch Direction:         0 degrees
       Wind Speed:        10 m/s
   Wind Direction:        45 degrees

            Range:   33.6821 meters
Landing Direction:   5.20345 degrees
         Air Time:     2.422 seconds
```

Case 3: 10 m/s Wind at 90 degrees from launch direction

```
     Launch Speed:        20 m/s
     Launch Angle:        40 degrees above horizontal
 Launch Direction:         0 degrees
       Wind Speed:        10 m/s
   Wind Direction:        90 degrees

            Range:   28.0928 meters
Landing Direction:   11.4065 degrees
         Air Time:     2.353 seconds
```

Case 4: 10 m/s Wind at 180 degrees from launch direction

```
     Launch Speed:        20 m/s
     Launch Angle:        40 degrees above horizontal
 Launch Direction:         0 degrees
       Wind Speed:        10 m/s
   Wind Direction:       180 degrees

            Range:   18.7246 meters
Landing Direction:         0 degrees
         Air Time:     2.289 seconds
```

Projectile Aiming Program

Test 1: Prediction Case 1

Mass:	5	kilograms
Drag Coefficient:	0.47	
Air Density:	1.225	kg/m^3
Cross-sectional Area:	0.25	m^2
Range:	28.6023	meters
Landing Direction:	0	degrees
Wind Speed:	0	m/s
Wind Direction:	0	degrees
Launch Speed:	19.9675	m/s
Launch Angle:	40	degrees above horizontal
Launch Direction:	0	degrees

Test 2: Prediction Case 2

Mass:	5	kilograms
Drag Coefficient:	0.47	
Air Density:	1.225	kg/m^3
Cross-sectional Area:	0.25	m^2
Range:	33.6703	meters
Landing Direction:	5.20345	degrees
Wind Speed:	10	m/s
Wind Direction:	45	degrees
Launch Speed:	19.99	m/s
Launch Angle:	40	degrees above horizontal
Launch Direction:	0.195124	degrees

Test 3: Prediction Case 3

Mass:	5	kilograms
Drag Coefficient:	0.47	
Air Density:	1.225	kg/m^3
Cross-sectional Area:	0.25	m^2
Range:	28.074	meters
Landing Direction:	11.4065	degrees
Wind Speed:	10	m/s
Wind Direction:	90	degrees
Launch Speed:	19.9754	m/s
Launch Angle:	40	degrees above horizontal

```
    Launch Direction:    0.22443 degrees
```

Test 4: Prediction Case 4

```
                 Mass:          5 kilograms
     Drag Coefficient:       0.47
          Air Density:      1.225 kg/m^3
Cross-sectional Area:       0.25 m^2

                Range:    18.6258 meters
   Landing Direction:          0 degrees
           Wind Speed:         10 m/s
      Wind Direction:        180 degrees

         Launch Speed:    19.9182 m/s
         Launch Angle:         40 degrees above horizontal
    Launch Direction:          0 degrees
```

Code

Projectile Simulation Header File:

```
#pragma once
#include <string>

using namespace std;

double deg_to_rad(double degrees);

double rad_to_deg(double radians);

void motion_calculation(double launch_angle, double initial_velocity, double launch_direction, double wind_speed, double wind_direction, double& ranges, double& landing_direction);

string output_parameters();
```

Projectile Simulation Function:

```
#include "VectorMath.h"
#include <vector>
```

```
#include <string>
#include <sstream>
#include <iomanip>

constexpr double pi = 3.14159265358979324;
constexpr double drag_c = 0.47;
constexpr double air_dense = 1.225;//kg/m^3
constexpr double x_area = 0.15;//m^2
constexpr double mass = 40;//kg

using namespace std;

//angle unit conversion functions
double deg_to_rad(double degrees)
{
	return pi * degrees / 180.0;
}
double rad_to_deg(double radians)
{
	return 180.0 * radians / pi;
}

void motion_calculation(double launch_angle, double launch_speed, double launch_direction, double wind_speed,
double wind_direction, double& range, double& landing_direction)
{
	//x-component is horizontal velocity along the original launch path
	//y-component is horizontal velocity perpendicular to original launch path
	//z-component is vertical velocity
	vector<double> velocity{ 0, 0, 0 };
	vector<double> position{ 0, 0, 0 };
	vector<double> wind{ 0, 0, 0 };
	vector<double> acceleration;

	velocity[0] = launch_speed * cos(deg_to_rad(launch_angle)) * cos(deg_to_rad(launch_direction));//sets
x-velocity m/s
```

```
	velocity[1] = launch_speed * cos(deg_to_rad(launch_angle)) * sin(deg_to_rad(launch_direction));//sets y-velocity m/s
	velocity[2] = launch_speed * sin(deg_to_rad(launch_angle));//sets z-velocity m/s
	double speed = launch_speed;//sets velocity

	wind[0] = wind_speed * cos(deg_to_rad(wind_direction));//sets x-velocity m/s
	wind[1] = wind_speed * sin(deg_to_rad(wind_direction));//sets y-velocity m/s

	double accel_drag; //initialization: how much speed lost due to drag

	double dt = 0.001; //time interval between calculations

	//simulates projectile motion in air
	do
	{
		//takes care of position change using velocity
		position = vector_add(position, scalar_multiplication(dt, velocity));

		//defines acceleration due to drag and gravity
		vector<double> wind_relative_velocity = vector_subtract(wind, velocity);
		accel_drag = 0.5 * drag_c * air_dense * x_area * pow(magnitude(wind_relative_velocity), 2) / mass;
		acceleration = scalar_multiplication(accel_drag, normalize(wind_relative_velocity));
		acceleration[2] -= 9.8;

		//changes velocity due to acceleration
		velocity = vector_add(velocity, scalar_multiplication(dt, acceleration));
	} while (position[2] > 0);//repeat while hasn't hit ground yet

	//calculates landing distance and direction
	//uses references to "return" multiple values
	position.pop_back();
	range = magnitude(position);
	landing_direction = rad_to_deg(atan2(position[1], position[0]));
}
```

```
string output_parameters()
{
	stringstream ss;
	ss << setw(22) << "Mass: " << setw(10) << mass << " kilograms" << endl
		<< setw(22) << "Drag Coefficient: " << setw(10) << drag_c << endl
		<< setw(22) << "Air Density: " << setw(10) << air_dense << " kg/m^3" << endl
		<< setw(22) << "Cross-sectional Area: " << setw(10) << x_area << " m^2" << endl
		<< endl;
	return ss.str();
}
```

Single Launch Test

```
#include "pch.h"
#include "VectorMath.h"
#include <vector>
#include <iostream>
#include <iomanip>
#include <sstream>

constexpr double pi = 3.14159265358979324;
constexpr double drag_c = 0.47;
constexpr double air_dense = 1.225;//kg/m^3
constexpr double x_area = 0.25;//m^2
constexpr double mass = 5.0;//kg

constexpr double launch_speed = 20.0;//m/s
constexpr double launch_angle = 40.0;//degrees
constexpr double launch_direction = 0.0;//degrees
constexpr double wind_speed = 10.0;//m/s
constexpr double wind_direction = 180.0;//degrees

using namespace std;

//angle unit conversion functions
double deg_to_rad(double degrees)
```

```
{
	double radians = pi * degrees / 180.0;
	while (radians >= 2 * pi) radians -= 2 * pi;
	while (radians < 0) radians += 2 * pi;
	return radians;
}
double rad_to_deg(double radians)
{
	double degrees = 180.0 * radians / pi;
	while (degrees >= 360.0) degrees -= 360.0;
	while (degrees < 0.0) degrees += 360.0;
	return degrees;
}

int main()
{
	//x-component is horizontal velocity along the original launch path
	//y-component is horizontal velocity perpendicular to original launch path
	//z-component is vertical velocity
	vector<double> velocity{ 0, 0, 0 };
	vector<double> position{ 0, 0, 0 };
	vector<double> wind{ 0, 0, 0 };
	vector<double> acceleration{ 0, 0, -9.8 };

	velocity[0] = launch_speed * cos(deg_to_rad(launch_angle)) * cos(deg_to_rad(launch_direction));//sets
x-velocity m/s
	velocity[1] = launch_speed * cos(deg_to_rad(launch_angle)) * sin(deg_to_rad(launch_direction));//sets
y-velocity m/s
	velocity[2] = launch_speed * sin(deg_to_rad(launch_angle));//sets z-velocity m/s
	double speed = launch_speed;//sets velocity

	wind[0] = wind_speed * cos(deg_to_rad(wind_direction));//sets x-velocity m/s
	wind[1] = wind_speed * sin(deg_to_rad(wind_direction));//sets y-velocity m/s

	double accel_drag; //initialization: how much speed lost due to drag
```

```
	double dt = 0.001;
	double air_time = 0;

	//loop simulates projectile motion in air
	do
	{
		//takes care of position change using velocity
		position = vector_add(position, scalar_multiplication(dt, velocity));
		air_time += dt;

		//defines acceleration due to drag and gravity
		vector<double> wind_relative_velocity = vector_subtract(wind, velocity);
		accel_drag = 0.5 * drag_c * air_dense * x_area * pow(magnitude(wind_relative_velocity), 2) /
mass;
		acceleration = scalar_multiplication(accel_drag, normalize(wind_relative_velocity));
		acceleration[2] -= 9.8;

		//changes velocity due to acceleration
		velocity = vector_add(velocity, scalar_multiplication(dt, acceleration));
	} while (position[2] > 0);//repeat while hasn't hit ground yet

	//calculates landing position
	position.pop_back();
	double range = magnitude(position);
	double landing_direction = rad_to_deg(atan2(position[1], position[0]));

	//prepare output data
	stringstream ss;
	ss << setw(22) << "Mass: " << setw(10) << mass << " kilograms" << endl
		<< setw(22) << "Drag Coefficient: " << setw(10) << drag_c << endl
		<< setw(22) << "Air Density: " << setw(10) << air_dense << " kg/m^3" << endl
		<< setw(22) << "Cross-sectional Area: " << setw(10) << x_area << " m^2" << endl
		<< endl
		<< setw(22) << "Launch Speed: " << setw(10) << launch_speed << " m/s" << endl
		<< setw(22) << "Launch Angle: " << setw(10) << launch_angle << " degrees above horizontal"
<< endl
```

```
        << setw(22) << "Launch Direction: " << setw(10) << launch_direction << " degrees" << endl
        << setw(22) << "Wind Speed: " << setw(10) << wind_speed << " m/s" << endl
        << setw(22) << "Wind Direction: " << setw(10) << wind_direction << " degrees" << endl
        << endl
        << setw(22) << "Range: " << setw(10) << range << " meters" << endl
        << setw(22) << "Landing Direction: " << setw(10) << landing_direction << " degrees" << endl
        << setw(22) << "Air Time: " << setw(10) << air_time << " seconds" << endl;
    cout << ss.str();
    return 0;
}
```

Aiming Program

```
#include "ProjectileMotion.h"
#include "VectorMath.h"
#include <iostream>
#include <iomanip>
#include <sstream>
#include <cmath>
#include <thread>
#include <utility>

constexpr double target_range = 100;//meters
constexpr double target_direction = 0;//degrees (measures from North, going clockwise)
constexpr double wind_speed = 0;//m/s
constexpr double wind_direction = 0;//degrees (measures from North, going clockwise)

using namespace std;

//calculates the distance between two points on a 2-D plane
double find_distance(pair<double, double> location, pair<double, double> target)
{
    return sqrt(pow(location.first - target.first, 2) + pow(location.second - target.second, 2));
}

//turns a polar coordinate into a rectangular coordinate
pair<double, double> polar_to_rect(double range, double direction)
```

```
{
	return make_pair(range * cos(deg_to_rad(direction)), range * sin(deg_to_rad(direction)));
}

int main()
{
	pair<double, double> target = make_pair(target_range*cos(deg_to_rad(target_direction)), target_range
*sin(deg_to_rad(target_direction)));

	double range, direction;
	double v_plus_range, v_plus_direction;
	double v_minus_range, v_minus_direction;
	double theta_plus_range, theta_plus_direction;
	double theta_minus_range, theta_minus_direction;

	double launch_speed = pow(target_range * 9.81, 0.5);
	double launch_angle = 35;
	double launch_direction = target_direction;

	do
	{
		//simultaneously calculate 5 slightly-different projectile motions
		thread t1(motion_calculation, launch_angle, launch_speed, launch_direction, wind_speed,
wind_direction, ref(range), ref(direction));
		thread t2(motion_calculation, launch_angle, launch_speed + 0.05, launch_direction, wind_speed,
wind_direction, ref(v_plus_range), ref(v_plus_direction));
		thread t3(motion_calculation, launch_angle, launch_speed - 0.05, launch_direction, wind_speed,
wind_direction, ref(v_minus_range), ref(v_minus_direction));
		thread t4(motion_calculation, launch_angle, launch_speed, launch_direction + 0.5, wind_speed,
wind_direction, ref(theta_plus_range), ref(theta_plus_direction));
		thread t5(motion_calculation, launch_angle, launch_speed, launch_direction - 0.5, wind_speed,
wind_direction, ref(theta_minus_range), ref(theta_minus_direction));
		if (t1.joinable()) t1.join();
		if (t2.joinable()) t2.join();
		if (t3.joinable()) t3.join();
		if (t4.joinable()) t4.join();
```

```
			if (t5.joinable()) t5.join();

			//if simulated motion is close enough to landing site
			if (find_distance(polar_to_rect(range, direction), target) <= 1) break;

			//derivatives of distance to landing site with respect to velocity and theta
			double dd_dv = (find_distance(polar_to_rect(v_plus_range, v_plus_direction), target) -
find_distance(polar_to_rect(v_minus_range, v_minus_direction), target)) / 0.1;
			double dd_dtheta = (find_distance(polar_to_rect(theta_plus_range, theta_plus_direction), target) -
find_distance(polar_to_rect(theta_minus_range, theta_minus_direction), target));

			//use gradient descent to adjust launch speed and launch direction
			launch_speed += dd_dv * (-0.01);
			launch_direction += dd_dtheta * (-0.1);
		} while (true);

		//prepares output information
		stringstream ss;
		ss << output_parameters()
			<< setw(22) << "Range: " << setw(10) << range << " meters" << endl
			<< setw(22) << "Landing Direction: " << setw(10) << target_direction << " degrees" << endl
			<< setw(22) << "Wind Speed: " << setw(10) << wind_speed << " m/s" << endl
			<< setw(22) << "Wind Direction: " << setw(10) << wind_direction << " degrees" << endl
			<< endl
			<< setw(22) << "Launch Speed: " << setw(10) << launch_speed << " m/s" << endl
			<< setw(22) << "Launch Angle: " << setw(10) << launch_angle << " degrees above horizontal"
<< endl
			<< setw(22) << "Launch Direction: " << setw(10) << launch_direction << " degrees" << endl;
		cout << ss.str();
   return 0;
}
```

Vector Math Functions

```
#pragma once
#include <vector>
```

```
using namespace std;

//vector addition and subtraction and multiplication
vector<double> vector_add(vector<double> vector1, vector<double> vector2)
{
	vector<double> result;
	for (int i = 0; i < vector1.size(); i++)
		result.push_back(vector1[i] + vector2[i]);
	return result;
}
vector<double> vector_subtract(vector<double> vector1, vector<double> vector2)
{
	vector<double> result;
	for (int i = 0; i < vector1.size(); i++)
		result.push_back(vector1[i] - vector2[i]);
	return result;
}
vector<double> scalar_multiplication(double scalar, vector<double> vector1)
{
	vector<double> result;
	for (int i = 0; i < vector1.size(); i++)
		result.push_back(vector1[i] * scalar);
	return result;
}

//returns magnitude of vector
double magnitude(vector<double> vector1)
{
	double square_sum = 0.0;
	for (int i = 0; i < vector1.size(); i++)
		square_sum += pow(vector1[i], 2);
	return pow(square_sum, 0.5);
}

//normalized vector (magnitude == 1)
vector<double> normalize(vector<double> vector1)
```

```
{
	vector<double> result;
	double size = magnitude(vector1);
	for (int i = 0; i < vector1.size(); i++)
		result.push_back(vector1[i] / size);
	return result;
}
```

Vector Math Header File

```
#pragma once
#include <vector>

using namespace std;

//vector addition and subtraction and multiplication
vector<double> vector_add(vector<double> vector1, vector<double> vector2);

vector<double> vector_subtract(vector<double> vector1, vector<double> vector2);

vector<double> scalar_multiplication(double scalar, vector<double> vector1);

//returns magnitude of vector
double magnitude(vector<double> vector1);

//normalized vector (magnitude == 1)
vector<double> normalize(vector<double> vector1);
```

Most Significant Achievement

Our most significant achievement was creating a program that has real world applications. While there were a number of ideas that we had that would have very few applications in the real world, this project provides a solution to a problem that is frequently faced in a number of applications throughout companies such as SpaceX with launches as well as through numerous military applications. We were also able to create a program that could be easily adapted to other problems, such as adding constant thrust to the model to make it able to predict the landing sites of missiles with constant acceleration or thrust, or could be modified to include the varying forces of gravity at different elevations so that it could be utilized to calculate the crash site for fuel containers on rocket launches along with a slew of other applications. Creating a program that is highly reliable, expandable, and applicable was our most significant achievement.

Conclusions

The results shown above are a compilation of a number of different test scenarios in which singular variables are changed to demonstrate different test scenarios. Each case has a labelled difference between that case and the control case, or in some cases another case that is specified. Each of these shows a number of things, including the optimal launch angle around 45 degrees, which is accurate, but shows the slight flaws in the program as the optimal launch angle is not truly 45 degrees in reality. Using the equations previously mentioned the program calculates to a relatively accurate degree the landing site of any projectile fired. In each of the tables there is a value marked as “website data”. This data is what was found from online calculators and other projection models, and is purely there for a reference to another model that is available.

Recommendations

As our project draws to a close, there are a number of areas that we were unable to address as a group that we believe future projects could probably end up improving on given more time, resources, etc. First, the scope of our project is decently small, focusing only on one specific aspect of physics. Although this area has numerous real world applications, the fact remains that our research area only explored a very small area of research. The simplest way to expand upon the scope of our project would have been to integrate it into some sort of specific real world scenario, instead of just simply launching an object at random. Second, a concept that we did not explore while developing our code was the idea of Reynold's number, which describes the motion of objects in liquids. In this case, air is considered to be a liquid, and Reynold's number would influence the way that projectiles travelled within it. Given the time and resources we had, we were unable to integrate Reynold's number into our code, but in the future very advanced code about projectile motion would have to incorporate this concept. Third, adding the feature of allowing different landing heights in our "projectile aiming" program would expand the usability and versatility of the project, but would complicate the process of choosing a launch angle. Incorporating the launch angle into our gradient descent calculations would produce a near infinite number of solutions. Without a practical way to automatically determine the launch angle, this option would be difficult to implement.

A simpler extension of our project includes changing our programs to treat air density and gravitational acceleration as variables rather than constants. Although the changes in these variables are negligible at ground level, adapting this program to greater altitudes (for example, for use in orbital predictions or rocket return landings) would require gravitational acceleration and air density to vary with altitude.

Acknowledgements

We would like to thank both Dennis Trujillo and Drew Einhorn for their feedback and contributions to our project and their excellent feedback. A number of the additions to our project came from their guidance and suggestions, and their contributions are greatly appreciated.