Computing the Trajectory of a Projectile

Josephine Monica **Aaron Khonstantine**
Nanyang Technological University, Singapore
E-mail: `M140075@e.ntu.edu.sg` `AA0001ne@e.ntu.edu.sg`

Abstract. We produce a program with C programming language for calculating trajectory distance of a particle thrown from a cliff. Program user can input various parameters such as launching angle, velocity, and initial height that will affect the trajectory distance. Several edge cases are handled to make the program robust. In this report, we present the theory, source code, description, sample runs of the program. Flowchart of the program is also included to illustrate logic flow of the program.

1. Theory

Consider a projectile being thrown from height y_o with initial velocity v at angle θ as shown in Figure 1. The projectile does not receive any external force in horizontal x component, so the

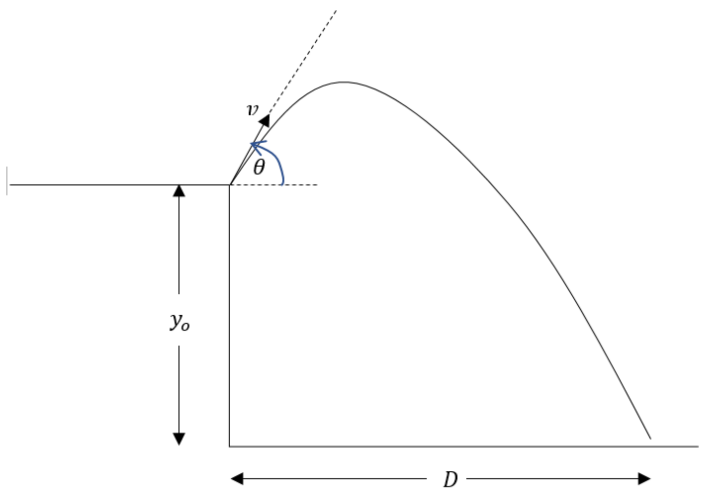

Figure 1: Trajectory of a projectile launched at with initial velocity v at angle θ from height y_o.

motion in the horizontal axis can be written as:

$$x = v\,cos(\theta) \times t \tag{1}$$

where x is the horizontal distance traveled by the projectile at time t. On the vertical axis, the projectile undergoes negative acceleration g by gravitational force. The motion of the projectile in the vertical component can be written as:

$$y = y_o + v\,sin(\theta) \times t - \frac{1}{2}\,g\,t^2 \tag{2}$$

where y is the vertical position of the projectile at time instance t.

The horizontal distance D is achieved when the projectile hits the ground, $y = 0$. By combining equation 1 and equation 2 and substituting $y = 0$ and $x = D$, we can the following quadratic equation of D:

$$0 = y_o + D\,tan(\theta) - \frac{g}{2v^2(cos(\theta))^2}D^2 \tag{3}$$

The quadratic equation can be easily solved giving 2 solutions of D:

$$D = \frac{-tan\theta \pm \sqrt{tan^2\theta + \frac{2\,g\,y_o}{v^2cos^2\theta}}}{\frac{-g}{v^2cos^2\theta}} \tag{4a}$$

However, note that D must be positive, hence we only consider the positive solution of D:

$$D = \frac{v^2}{g}\left[sin\theta\,cos\theta + \sqrt{sin^2\theta\,cos^2\theta + \frac{2\,g\,y_o\,cos^2\theta}{v^2}}\right] \tag{4b}$$

We define the range of several parameters in equation 4b :

- Initial height: $y_o \geqslant 0$
  The initial height is defined to be positive. Negative initial height results to different problem in which the projectile will collide with the wall when launched at e.g. angle of 30^o.
- Initial velocity $v \geqslant 0$
  Similarly, initial velocity is defined to be positive. Negative initial height results to different problem. To illustrate this, consider a case where the projectile is launched with negative velocity $-1m/s$ at angle 45^o. In this case, the projectile will not be launched to the air, instead it will hit the cliff as soon as it is launched.
- Launching angle $-\frac{\pi}{2} \leqslant \theta \leqslant \frac{\pi}{2}$
  The launching angle is specified to be in the 1^{st} and 4^{th} quadrants. Launching the projectile at angle in the 3^{rd} does not set the projectile to the air, but it will hit the cliff as soon as it is launched. On the other hand, launching the projectile at angle in the 2^{nd} quadrant results to different problem: the projectile will land at $y = y_o$ as opposed to $y = 0$, so equation 4b does not hold in this case.

We will use these parameters' ranges in our program implementation.

2. Program Description

The use of the program is to compute the horizontal distance D of a projectile thrown with initial velocity v at launching angle θ from a cliff of height y_o as shown in Figure 1. There are 3 available modes which user can choose to run the program:

- Mode 1: Computing D for different values of launching angle in range θ_1 to θ_2 , with other parameters (v and y_o) being constant.
- Mode 2: Computing D for different values of initial velocity in range v_1 to v_2 , with other parameters (θ and y_o) being constant.
- Mode 3: Computing D for different values of initial height in range y_1 to y_2 , with other parameters (v and θ) being constant.
- User can exit the program by inputting 0 to the mode.

For every user input, the program also does parameters check to the user inputs. If the user enters input that is not in the range of valid parameter (as discussed in section 1), the program will give error message and re-ask the user for valid input. User will also specify the number of output data points (N). For each mode, the program will compute D for N different values of varying parameter with uniform increment. For example, in mode 1 the program will compute D for different values of θ : $\{\theta_1,\ \theta_1 + \frac{\theta_2-\theta_1}{N-1},\ \theta_1 + 2\frac{\theta_2-\theta_1}{N-1},\ ...,\ \theta_2\}$. Once the program displays its output, it will again prompt the user to enter mode. User can continue the program or exit by specifying 0 as the mode.

3. Flowchart

We present the flowchart of the program. Figure 2 shows the overall flowchart of the program. It has three sub-programs called "angle varies", "velocity varies", and "height varies". The flowcharts' of these sub-programs are illustrated in Figure 3. ,Figure 4. ,Figure 5. respectively.

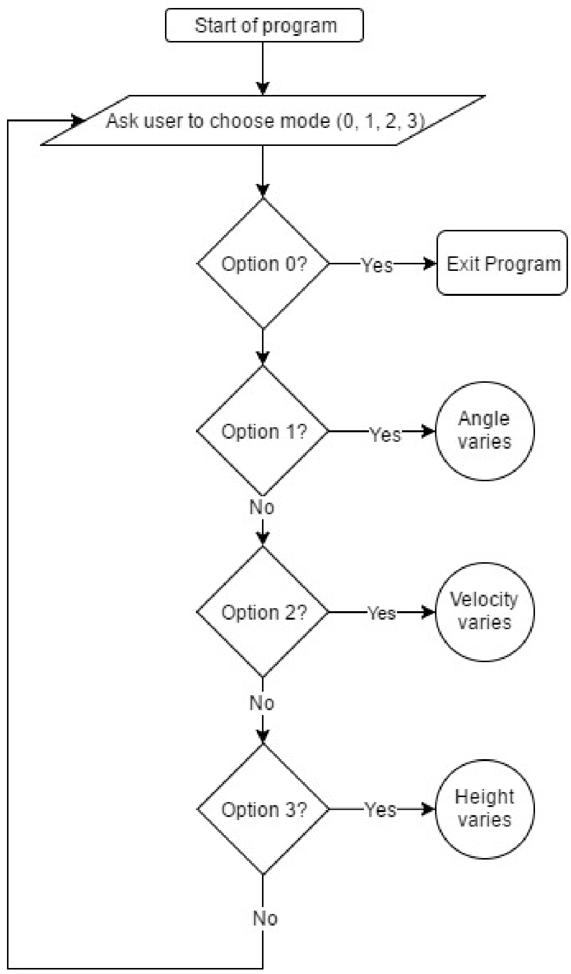

Figure 2: Overall flowchart of the program.

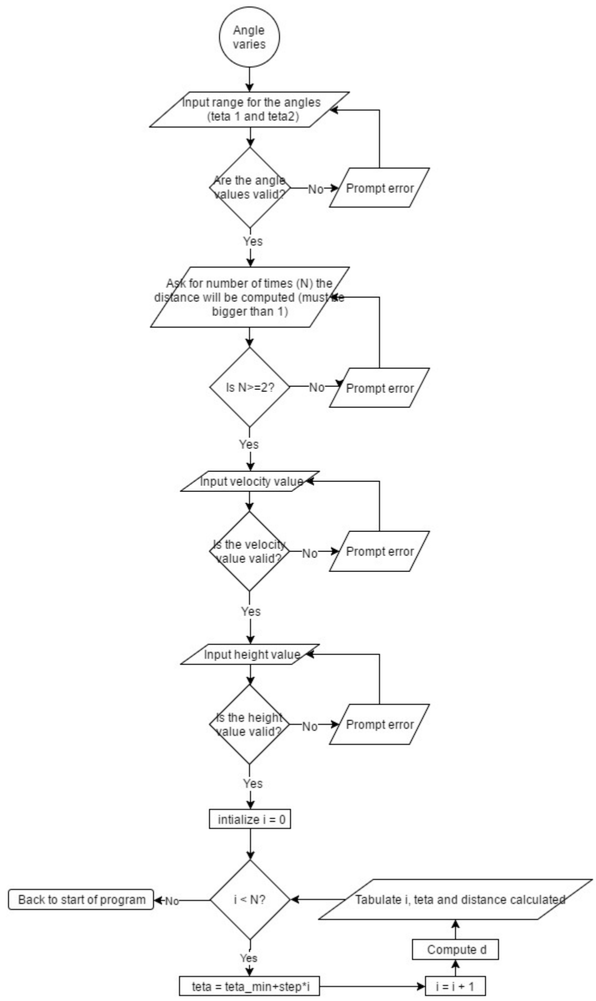

Figure 3: Sub-flowchart for mode 1 (angle varies).

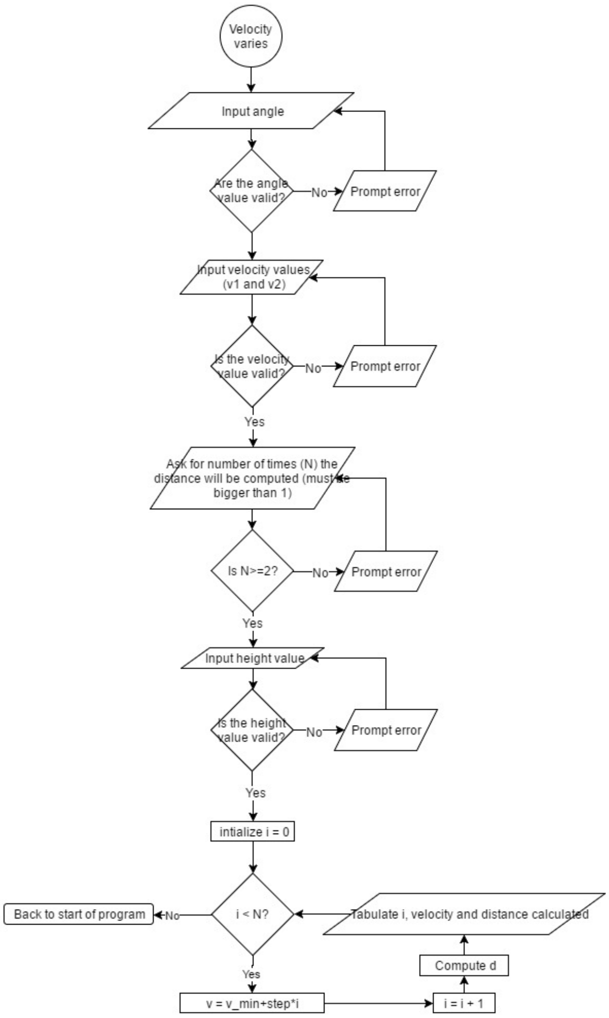

Figure 4:
Sub-flowchart for mode 2 (velocity varies).

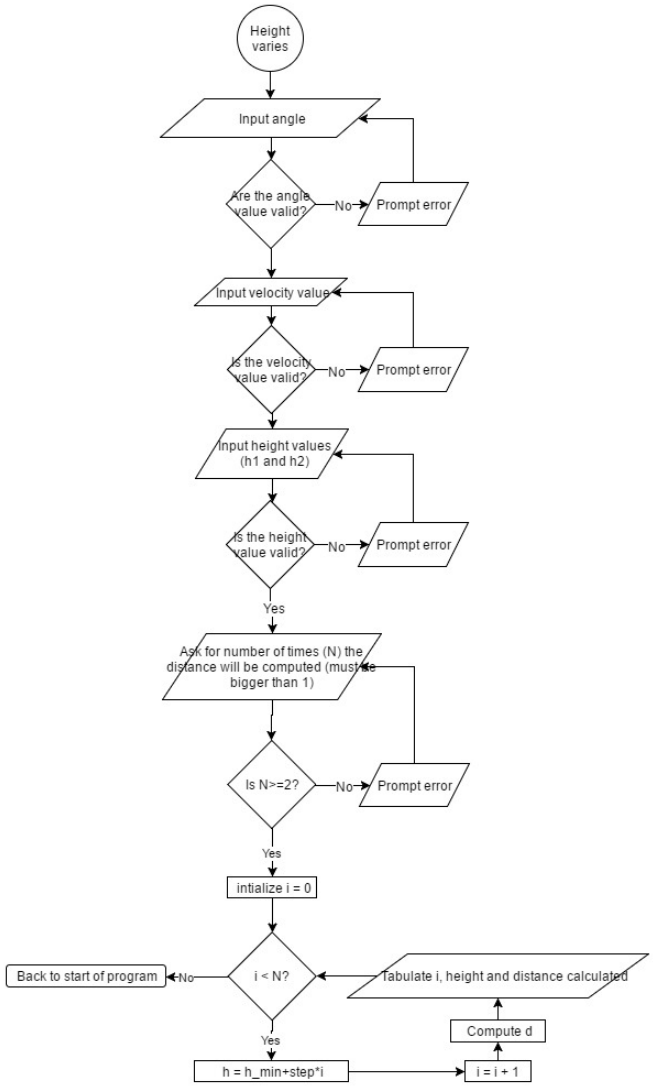

Figure 5:
Sub-flowchart for mode 3 (height varies).

4. Source Code

We present the source code of the program "projectile.c". To compile the code using `gcc`, one can run the following command:

```
# gcc -o projectile projectile.c -lm
```

This will create executable program called `projectile` which can be launched by simply calling:

```
# ./projectile
```

```c
/**
    MA4830 Minor Assignment
    projectile.c
    Purpose: Compute the trajectory distance of a projectile.
             Parameters include initial velocity, launching angle, and initial height.
             User can choose to vary one of this parameter, and the program will tabulate the trajectory distance
    @author Josephine Monica, Aaron Khonstantine
    @version 1 11/10/17
*/

#include <stdio.h>
#include <math.h>

#define g 9.81    //Macro for gravitational value
#define pi 3.1415 //Macro for pi

/*********************function prototypes*******************/
/**
    Recursively prompt user to enter input mode (0,1,2,3) of the program until valid mode is entered

    @param mode_: The user input mode will be stored in this parameter
    @return void
*/
void input_mode(int* mode_);

/**
    Recursively prompt user to enter angle value until valid angle (-pi/2 to pi/2) is entered

    @param teta_: The user input angle will be stored in this parameter
    @return void
*/
void input_angle(double* teta_);

/**
    Recursively prompt user to enter velocity value until valid velocity (v>=0) is entered

    @param v: The user input velocity will be stored in this parameter
    @return void
*/
void input_velocity(double* v_);

/**
```

```
    Recursively prompt user to enter height value until valid height (y>=0) is entered

    @param y_: The user input height will be stored in this parameter
    @return void
*/
void input_y(double* y_);

/**
    Recursively prompt user to enter N_ (number of output data points) value
    until valid N_ (at least 2 data points) is entered

    @param N_: The user input N_ will be stored in this parameter
    @return void
*/
void input_N(int* N_);

/**
    Compute and return trajectory distance of projectile
    launched with initial velocity v_ at angle teta_ and at initial height of y_

    @param teta_: launching angle
    @param v_  : initial velocity
    @param y_  : initial height
    @return The computed trajectory distance for given parameters
*/
double compute_d(double teta_, double v_, double y_);

/****************************************************/
int main()
{
    //variables
    int mode;                    //mode of program (vary v / theta/ height)
    double v1,v2,teta1,teta2,y1,y2;
    int N;                       //Number of data points that the program will output

    //auxillary variables
    int i;double step;

    while(1)                     //repeat the program until the user chooses to exit
    {
        //ask the user to choose mode (1: vary theta, 2: vary velocity, 3: vary height, or 0 :EXIT the program)
        input_mode(&mode);

        switch (mode)
        {
            //EXIT the program
            case 0:
                return 0;

            //vary theta
            case 1:
                //take necessary input parameters
                printf("Enter teta1\n");        //angle range
```

```c
            input_angle(&teta1);
            printf("Enter teta2\n");
            input_angle(&teta2);

            input_N(&N);                    //N

            input_velocity(&v1);            //velocity

            input_y(&y1);                   //height

            //print the overhead of the table
            printf("\nCOMPUTING d for teta= %lf to %lf , v= %lf, y=%lf\n",teta1,teta2,v1,y1);
            printf("----------------------------------------------------\n");
            printf("No\t teta \t\t\t d\n");
            printf("----------------------------------------------------\n");

            double teta_min,teta_max;
            teta_min=fmin(teta1,teta2);
            teta_max=fmax(teta1,teta2);
            step=(teta_max-teta_min)/(N-1);

            //compute and print the outputs (N data points for varying theta: teta_min to teta_max)
            for(i=0;i<N;i++)
            {
                printf("%d\t %lf \t\t %lf\n",i+1,teta_min+i*step,compute_d(teta_min+i*step,v1,y1));
            }
            break;

        //vary v
        case 2:
            //take necessary input parameters
            input_angle(&teta1);            //angle

            printf("Enter v1\n");           //velocity range
            input_velocity(&v1);
            printf("Enter v2\n");
            input_velocity(&v2);

            input_N(&N);                    //N

            input_y(&y1);                   //height

            //print the overhead of the table
            printf("\nCOMPUTING d for teta= %lf, v= %lf to %lf , y=%lf\n",teta1,v1,v2,y1);
            printf("----------------------------------------------------\n");
            printf("No\t v \t\t\t d\n");
            printf("----------------------------------------------------\n");

            double v_min,v_max;
            v_min=fmin(v1,v2);
```

```c
                v_max=fmax(v1,v2);
                step=(v_max-v_min)/(N-1);

                //compute and print the outputs (N data points for varying v: v_min to v_max)
                for(i=0;i<N;i++)
                {
                    printf("%d\t %lf \t\t %lf\n",i+1,v_min+i*step,compute_d(teta1,v_min+i*step,y1));
                }
                break;

            //vary y
            case 3:
                //take necessary input parameters
                input_angle(&teta1);            //angle

                input_velocity(&v1);            //velocity

                printf("Enter y1\n");           //initial height range
                input_y(&y1);
                printf("Enter y2\n");
                input_y(&y2);

                input_N(&N);                    //N

                //print the overhead of the table
                printf("\nCOMPUTING d for teta= %lf, v= %lf , y=%lf to %lf \n",teta1,v1,y1,y2);
                printf("-----------------------------------------------------\n");
                printf("No\t y \t\t\t d\n");
                printf("-----------------------------------------------------\n");

                double y_min,y_max;
                y_min=fmin(y1,y2);
                y_max=fmax(y1,y2);
                step=(y_max-y_min)/(N-1);

                //compute and print the outputs (N data points for varying height: y_min to y_max)
                for(i=0;i<N;i++)
                {
                    printf("%d\t %lf \t\t %lf\n",i+1,y_min+i*step,compute_d(teta1,v1,y_min+i*step));
                }

                break;
        }
    }
}
/*****************************************************/

//ask user to input mode
void input_mode(int* mode_)
```

```c
{

    //input mode
    printf("\n>>Enter your mode selection:\n");
    printf("1 for varying angle\n");
    printf("2 for varying velocity\n");
    printf("3 for varying initial height\n");
    printf("0 to exit the program\n");

    int s; //scanf return value
    s=scanf("%d",mode_);
    while ((getchar()) != '\n'); //flush the scanf buffer. This is especially useful when the argument is not of correct type

    if(s==0)   ///re-ask user if input is not of type integer
    {
        printf("ERR: Invalid argument. Input is not an integer.\n");
        input_mode(mode_);
    }
    else       //check if input is valid (0,1,2,3)
    {
        switch(*mode_){
            case 1:
                printf("mode 1: Variation in ANGLE from teta1 to teta2\n\n");
                break;
            case 2:
                printf("mode 2: Variation in VELOCITY from v1 to v2\n\n");
                break;
            case 3:
                printf("mode 3: Variation in HEIGHT from y1 to y2\n\n");
                break;
            case 0:
                printf("EXIT the program\n\n");
                break;
            default:   //invalid input
                printf("ERR: Invalid argument. Please enter 0, 1, 2, or 3 \n");
                input_mode(mode_);    //ask the user to reenter mode
        }
    }
}

//ask user to input initial angle
void input_angle(double* teta_)
{
    //teta only from -90degree to 90degree, otherwise different computation
    printf(">>Enter angle (radian) from -pi/2 to pi/2: ");

    int s;
    s=scanf("%lf",teta_);
    while ((getchar()) != '\n'); //flush the scanf buffer. This is especially useful when the argument is not of the correct type

    if(s==0)                      ///re-ask user if input is not of type double
    {
```

```
        printf("ERR: Invalid argument. Input is not a double type.\n");
        input_angle(teta_);
    }
    else if(*teta_>pi/2 || *teta_<-pi/2) //re-ask user if input is not in valid range (-pi/2 to pi/2)
    {
        printf("ERR: Invalid argument. Angle must be between -pi/2 and pi/2\n");
        input_angle(teta_);
    }

}

//ask user to input initial velocity
void input_velocity(double* v_)
{
    //velocity must be at least 0 or positive
    printf(">>Enter positive velocity (m/s): ");

    int s;
    s=scanf("%lf",v_);
    while ((getchar()) != '\n'); //flush the scanf buffer. This is especially useful when the argument is not of the correct type

    if(s==0)            //re-ask user if input is not of type double
    {
        printf("ERR: Invalid argument. Input is not a double type.\n");
        input_velocity(v_);
    }
    else if(*v_<0)      //re-ask user if input is not in valid range (>0)
    {
        printf("ERR: Invalid argument. Velocity must be zero or positive\n");
        input_velocity(v_);
    }
}

//ask user to input initial height y
void input_y(double* y_)
{
    //y must be at least 0 or positive
    printf(">>Enter positive initial height (m): ");

    int s;
    s=scanf("%lf",y_);
    while ((getchar()) != '\n'); //flush the scanf buffer. This is especially useful when the argument is not of the correct type

    if(s==0)          //re-ask user if input is not of type double
    {
        printf("ERR: Invalid argument. Input is not a double type.\n");
        input_y(y_);
    }
    else if(*y_<0)    //re-ask user if input is not in valid range (>0)
    {
        printf("ERR: Invalid argument. Initial height must be zero or positive\n");
```

```c
        input_y(y_);
    }
}

//ask user to input N
void input_N(int* N_)
{
    //input N
    printf("d will be computed N times for different values of parameter in the
        range\n");
    printf("Note that minimum of N is 2 (to at least include the two boundaries in
        range) \n");
    printf(">>Enter desired N: ");

    int s; //scanf return value
    s=scanf("%d",N_);
    while ((getchar()) != '\n'); //flush the scanf buffer. This is especially useful
        when the argument is not of the correct type

    if(s==0)   //re-ask user if input is not of type integer
    {
        printf("ERR: Invalid argument. Input is not an integer.\n");
        input_N(N_);
    }
    else if(*N_<2) //re-ask user if input is not in valid range (at least 2)
    {
        printf("ERR: Invalid argument. N must be at least 2\n");
        input_N(N_);
    }
}

//returns computed trajectory distance
double compute_d(double teta_, double v_, double y_)
{
    /*INSTEAD of (pow(v_,2)*sin(2*teta_)/(2*g)) * (1 +
        sqrt(1+2*g*y_/(pow(v_,2)*pow(sin(teta_),2))))
    rewrite the formula as follos to avoid edge case division by 0
    and to allow correct value of for negative angle*/
    return (pow(v_,2)/(2*g)) * (sin(2*teta_) + sqrt( pow(sin(2*teta_),2) + 8*g*y_*
        pow(cos(teta_),2) /(pow(v_,2))));
}
```

5. Sample Run

In this section, we show various sample runs of the program. This include sample runs for each mode and sample runs for various invalid inputs.

- Figure 6. shows the sample run for mode 1 (varying angle). First, the programs asks the user to enter the desired mode. Once the user inputs mode 1, the program then runs in mode 1. It asks the user to input various parameters: range for angle variation (θ_1 to θ_2), number of data outputs (N), initial velocity (v), and initial height (y). The program then computes the horizontal distances for different angles and displays it in table. Once it is done, user can continue to use the program by selecting new mode or exit the program.

```
>>Enter your mode selection:
1 for varying angle
2 for varying velocity
3 for varying initial height
0 to exit the program
1
mode 1: Variation in ANGLE from teta1 to teta2

Enter teta1
>>Enter angle (radian) from -pi/2 to pi/2: 0.5
Enter teta2
>>Enter angle (radian) from -pi/2 to pi/2: 1.0
d will be computed N times for different values of parameter in the range
Note that minimum of N is 2 (to at least include the two boundaries in range)
>>Enter desired N: 6
>>Enter positive velocity (m/s): 7
>>Enter positive initial height (m): 1

COMPUTING d for teta= 0.500000 to 1.000000 , v= 7.000000, y=1.000000
----------------------------------------------------
No       teta                    d
----------------------------------------------------
1        0.500000                5.581490
2        0.600000                5.823883
3        0.700000                5.910888
4        0.800000                5.825200
5        0.900000                5.558692
6        1.000000                5.112299

>>Enter your mode selection:
1 for varying angle
2 for varying velocity
3 for varying initial height
0 to exit the program
```

Figure 6: Sample run of mode 1, varying angle.

- Figure 7. shows the sample run for mode 2 (varying velocity). First, the programs asks the user to enter the desired mode. Once the user inputs mode 2, the program then runs in mode 2. It asks the user to input various parameters: launching angle (θ), range for velocity variation (v_1 to v_2), number of data outputs (N), and initial height (y). The program then computes the horizontal distances for different initial velocities and displays it in table. Once it is done, user can continue to use the program by selecting new mode or exit the program.

```
>>Enter your mode selection:
1 for varying angle
2 for varying velocity
3 for varying initial height
0 to exit the program
2
mode 2: Variation in VELOCITY from v1 to v2

>>Enter angle (radian) from -pi/2 to pi/2: 0.5
Enter v1
>>Enter positive velocity (m/s): 3
Enter v2
>>Enter positive velocity (m/s): 9
d will be computed N times for different values of parameter in the range
Note that minimum of N is 2 (to at least include the two boundaries in range)
>>Enter desired N: 5
>>Enter positive initial height (m): 30

COMPUTING d for teta= 0.500000, v= 3.000000 to 9.000000 , y=30.000000
----------------------------------------------------
No       v                        d
----------------------------------------------------
1        3.000000                 6.908467
2        4.500000                 10.673590
3        6.000000                 14.657276
4        7.500000                 18.867877
5        9.000000                 23.313600

>>Enter your mode selection:
1 for varying angle
2 for varying velocity
3 for varying initial height
0 to exit the program
```

Figure 7: Sample run of mode 2, varying velocity.

- Figure 8. shows the sample run for mode 3 (varying height). First, the programs asks the user to enter the desired mode. Once the user inputs mode 3, the program then runs in mode 3. It asks the user to input various parameters: launching angle (θ), initial velocity (v), range for initial height (y_1 to y_2), and number of data outputs (N). The program then computes the horizontal distances for different initial velocities and displays it in table. Once it is done, user can continue to use the program by selecting new mode or exit the program.

```
>>Enter your mode selection:
1 for varying angle
2 for varying velocity
3 for varying initial height
0 to exit the program
3
mode 3: Variation in HEIGHT from y1 to y2

>>Enter angle (radian) from -pi/2 to pi/2: -0.1
>>Enter positive velocity (m/s): 3
Enter y1
>>Enter positive initial height (m): 0
Enter y2
>>Enter positive initial height (m): 15.6
d will be computed N times for different values of parameter in the range
Note that minimum of N is 2 (to at least include the two boundaries in range)
>>Enter desired N: 8

COMPUTING d for teta= -0.100000, v= 3.000000 , y=0.000000 to 15.600000
---------------------------------------------------
No       y                      d
---------------------------------------------------
1        0.000000               0.000000
2        2.228571               1.922985
3        4.457143               2.755802
4        6.685714               3.395041
5        8.914286               3.934010
6        11.142857              4.408883
7        13.371429              4.838219
8        15.600000              5.233045

>>Enter your mode selection:
1 for varying angle
2 for varying velocity
3 for varying initial height
0 to exit the program
```

Figure 8: Sample run of mode 3, varying height.

- Figure 9. shows sample run of user exiting the program. User can exit the program by inputting 0 in the mode selection.

```
>>Enter your mode selection:
1 for varying angle
2 for varying velocity
3 for varying initial height
0 to exit the program
0
EXIT the program

mon@mon:~/MA4830/mini_assignment$
```

Figure 9: User exits the program.

- Figure 10. shows sample run for invalid input of N. If user input is not of the correct data type, the program will print error message suggesting the user to input integer. If the user input is integer, but not in valid range of N, the program will print different error message suggesting the valid range of N for the user. It will then ask the user to re-input N value.

```
>>Enter desired N: abc
ERR: Invalid argument. Input is not an integer.
d will be computed N times for different values of parameter in the range
Note that minimum of N is 2 (to at least include the two boundaries in range)
>>Enter desired N: 1
ERR: Invalid argument. N must be at least 2
d will be computed N times for different values of parameter in the range
Note that minimum of N is 2 (to at least include the two boundaries in range)
>>Enter desired N: 4
>>Enter positive velocity (m/s):
```

Figure 10: Invalid N input.

- Figure 11. shows sample run for invalid input of θ. If user input is not of the correct data type, the program will print error message suggesting the user that the correct input is of double data type. If the user input is of double type, but not in valid range of θ, the program will print different error message suggesting the valid range of θ for the user. It will then ask the user to re-input θ value.

```
>>Enter angle (radian) from -pi/2 to pi/2: ma4
ERR: Invalid argument. Input is not a double type.
>>Enter angle (radian) from -pi/2 to pi/2: 10
ERR: Invalid argument. Angle must be between -pi/2 and pi/2
>>Enter angle (radian) from -pi/2 to pi/2: -1
Enter v1
>>Enter positive velocity (m/s):
```

Figure 11: Invalid angle input.

- Figure 12. shows sample run for invalid input of v. If user input is not of the correct data type, the program will print error message suggesting the user that the correct input is of double data type. If the user input is of double type, but not in valid range of v, the program will print different error message suggesting the valid range of v for the user. It will then ask the user to re-input v value.

```
>>Enter positive velocity (m/s): monic
ERR: Invalid argument. Input is not a double type.
>>Enter positive velocity (m/s): -1.1
ERR: Invalid argument. Velocity must be zero or positive
>>Enter positive velocity (m/s): 10
>>Enter positive initial height (m):
```

Figure 12: Invalid velocity input.

- Figure 13. shows sample run for invalid input of initial height y. If user input is not of the correct data type, the program will print error message suggesting the user that the correct input is of double data type. If the user input is of double type, but not in valid range of initial height y, the program will print different error message suggesting the valid range of v for the user. It will then ask the user to re-input the initial height y value.

```
>>Enter positive initial height (m): ghj
ERR: Invalid argument. Input is not a double type.
>>Enter positive initial height (m): -1
ERR: Invalid argument. Initial height must be zero or positive
>>Enter positive initial height (m): 10

COMPUTING d for teta= -1.000000, v= 1.000000 to 2.000000 , y=10.000000
----------------------------------------------------
No        v                        d
```

Figure 13: Invalid initial height input.