Laboratory 10 – Week 17

## Parameter Estimation - Projectile Modelling

## 10.1 Introduction

**Firstly, this worksheet is one of the worksheets from which your laboratory worksheets portfolio of work will be assessed.**

It is **VERY** important that you have completed Laboratory Sheet 9 - Hill Climbing - the Scales Problem. You should work on laboratory 9 and complete it before you start this one (sheet 10).

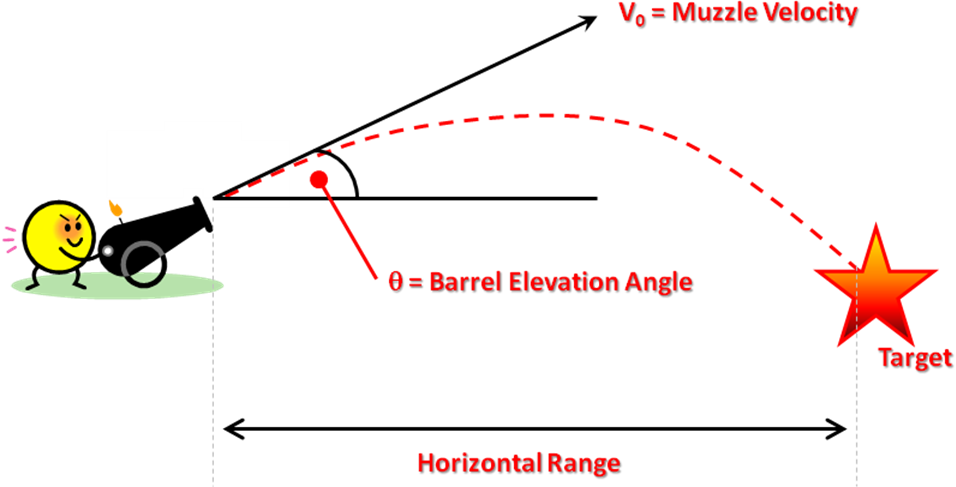
The laboratory involves the use and application of the Hill Climbing algorithm to solve a parameter estimation problem. You will be running a number of experiments and plotting convergence graphs.

## 10.2 Preliminaries

Familiarise yourself with the lecture entitled “9.1 Heuristic Search, Hill Climbing and Simulated Annealing”. Pay particular attention to the section on the Hill Climbing algorithm (The Random Mutation version - RMHC). Also read up on the lecture on applications and parameter estimation.

The parameter estimation problem we are solving is that of a cannon trying to hit a target. The cannon we are modelling is called K12 was built in 1938 and weighed over 300 tons. It had a very, very long range, and was an experimental prototype that was never really used. The problem with K12 was that it was ludicrously inaccurate, missing the intended target by sometimes up to several kilometres!

The reason for this is that at the extreme ranges that K12 could attain, the projectile reaches such a height that the air density is less and thus the drag (air resistance) is reduced. Modelling variable air resistance and drag is not a trivial problem, the standard Newtonian laws of motion cannot be used, and there is no algebraic solution to the problem - it is NP-Complete! The K12 engineers did not have access to computer models or the techniques that we have been learning about and thus the high levels of inaccuracy.



In order to hit a target two parameters have to be selected, one is barrel elevation angle (θ ranging between 25 and 55 degrees for K12), and the second is muzzle velocity (V0, in metres per second ranging from 1,500 to 1,650 for K12). This latter parameter is how fast the projectile leaves the barrel. The barrel elevation is measured from the horizontal plane/axis.

## 10.3 Exercise 1: The Cannon Class

In Appendix A there is a class that can be extracted, called Cannon.java.

This class has three methods (there are more but we will be using only three):

**public** **static** Double GetMaxRange(Double angle,Double muzzlevelocity)

This method simulates a K12 projectile being fired at a given angle (in degrees between 25-55) and muzzle velocity (in metres per second between 1,500-1,650). The return value is the horizontal range in metres. Note that these two parameters are restricted as described in the previous section.

**public** **static** ArrayList<Double> GetX()

This method returns an array list containing the simulated x-axis points the projectile passed through from the last call to GetMaxRange.

**public** **static** ArrayList<Double> GetY()

This method returns an array list containing the simulated y-axis points the projectile passed through from the last call to GetMaxRange.

If you get both the x-axis and y-axis points, then you can plot the flight of the K12 projectile. The length of the return values of GetX and GetY should always be the same as each other (array list size). Tell Dr Swift if it isn't!

Implement the follow code fragment:

**double** r = Cannon.*GetMaxRange*(40.0,1575.0);

System.*out*.println(r);

ArrayList<Double> xt = Cannon.*GetX*();

ArrayList<Double> yt = Cannon.*GetY*();

System.*out*.println(xt.size());

System.*out*.println(yt.size());

This simulates the cannon being fired at an angle of 40 degrees with a muzzle velocity of 1,575 metres per second. Print out (display or write to a file) the paired values for *x* and *y* and then plot them in Excel.

## 10.4 Exercise 2: Hill Climbing Method

We are now going to design a hill climbing algorithm to solve the following problems:

1. What is the maximum range of K12? What angle and muzzle velocity is needed?
2. What is the minimum range of K12? What angle and muzzle velocity is needed?
3. What angle and muzzle velocity is needed to hit a target 75,000 metres away? How close can you get?
4. What angle and muzzle velocity is needed to hit a target 95,000 metres away? How close can you get?
5. What angle and muzzle velocity is needed to hit a target 65,000 metres away? How close can you get?

You can reuse some of your previous hill climbing code for these exercises.

We will need to design/decide upon the following elements:

1. A representation
2. A fitness function
3. A random starting point
4. A small change operator
5. The number of iterations to run for

### 10.4.1 Representation

This is very simple, we represent both the angle (θ) and muzzle velocity (V0) as doubles. Exactly how you do this is down to your own design decisions, but it might be worth creating a CannonSolution class along the same lines as the ScalesSolution class used in the previous laboratories.

### 10.4.2 The Fitness Function

For exercise 2.1 (maximum range) we simply use the GetMaxRange method as the fitness and treat the problem as a maximisation problem, similarly for exercise 2.2 (minimum range) we treat the problem as minimisation problem.

For exercises 2.3, 2.4 and 2.5 (hitting a target at a specific range) we could use the following as a fitness function:

**double** range = Cannon.*GetMaxRange*(*theta*, *vzero*);

fitness = Math.*abs*(range-TargetRange);

Where TargetRange is either 75,000, 95,000 or 65,000 depending on what exercise we are solving.

### 10.4.3 Random Starting Point

We can generate a random angle (θ) between 25 and 55 and a random muzzle velocity (V0) between 1,500 and 1,650 using the random number generators from previous laboratories.

### 10.4.4 Small Change Operator

We have two real valued parameters we are trying to change. The suggestion for this is that you randomly decide which one you are going to change each time you come to change a parameter, i.e. you simulate the toss of a coin each time to decide which one we are going to change. Note that you will almost definitely have to optimise (determine) both parameters.

Once you have decided which parameter you are going to change, we need to alter them in a small manner. The suggestion is that we generate a real number between ±*x* where *x* is a small value and then add this value on to θ or V0 as appropriate. This value might not be the same for the two parameters. If it is too high then we will end up performing a random search, if it is too low then we might never find the optimal solution. Experiment with different values of *x*. A good starting value might be 1% of the difference between the maximum and minimum values for each parameter.

Once you have changed a parameter, you must check that it has not put the parameter out of range, i.e. exceeded the minimum or maximum possible values. A suggestion of how to cope with this is to check if the new value of the parameter has exceeded either the maximum or minimum and if this is the case, set it back to the maximum or minimum value as appropriate.

### 10.4.5 Iterations

Conduct a series of experiments to find the best number of iterations to run the hill climbing algorithm for.

### 10.4.6 Experiments

Plot convergence graphs for all of the experiments. Make sure you save any results and graphs for the viva.

## 10.5 Appendix A

The following class contains the code you will need for this exercise sheet:

|  |
| --- |
|  |