528649 Ismail bhamjee

Simulation and intelligent tracking of a robot

Introduction

The following documentation is about simulating a robot in 1D space, creating it a real-life scenario by adding noise which is always prevenient this it to add realism to the robot.

The noise that will be added will consist of number with zero mean and a standard deviation. These will be randomised to simulate uncertainty. In order to generate these number, we will use a box Muller method then we will add them to x (t) location (time).

In part three an intelligent agent will need to be designed which predicts the location of the robot from part one. This will require using the sensor readings from parts one and two. The agent will have a single neuron which will learn the moments of the first robot.

Part 1

In this part we generate numbers which represent the robot moving within the bounds of 2 and –2 this was the location of the robot.

The logic for changing U is found in the while loop with the addition of IF statements to find the correct values, between lines 43 and 63 . This is done via implementing the formula of X= -2X + 2U. The result will then be placed in an excel file for reading.

U will be plotted against time and it will change according to time. The graphs show that if we reduce the time step the values become less volatile, this is because Once x has gone to zero –2x will always remain zero this is called convergence. This shows the lower H the better values we will get. Unstable values of H, if we changed the value of H to 1.6 which is the upper limit and causes the program to lag out, however the lower limit is 0.0001 which also causes the program to cause an out of bounds error.

Euler's method the time derivate the difference between 2 values of x divided by the time to move between one value to the next, X is don’t dependant on the value of the h. this may lead to oscillations or large errors.

Part 2

The primary reason why noise was introduced was that we want to simulate real like errors such as: Implementation errors which are to do with the logic used such as step size. the smaller the step, the smaller the error, the greater the step the greater the error, modelling errors which are things like the environment used and how we didn’t take into consideration real world problems such as friction and drag, even these are approximations as they aren't what happens in the real world. Data errors these would be what data is collected by the robot using its hypothetical senses. Such as sensor inaccuracy and mechanical latency.

The noise we added was minuscule and did not affect the data set by much. Normally distributed is nose which is applied to all parts on the dataset not just a few. This is to account for errors all the way through the system. This shown via the graphs in part 2 as the line for noise existed but it was not visible, due to it being very similar to the original X value. This was done via reading the values of the first part from the excel fine and then adding noise to it to the existing values.

We generated 1 random number and the machine returned 2 this is because a positive version and a negative version were returned from the input given. As shown by X= -2X + 2U. We used the cos and sin functions which are the inverse of each other, so the mean was around 0, this allowed us to choose the standard deviation when calculating the noise.

The code in part 2 shows how the process of generating the random number within the bounds and adding it.

Part 3

In this part we had to generate a random number between –0.5 and 0.5 this would act as the weight the perceptron would use on top of the 3 past samples so that the learning algorithm would have enough data to work out the 4^{th}, as it is a single layered perceptron it would require more inputs. With these inputs it must be able to generate an output. This is called a feed forward network

The stepper function would be viable if the data was linearity separated, however the stepper can only return a Boolean value where as a sigmoid returns a number which can be used to see how off the prediction was as well as learn from it the rate it was off by.

Logistic sigmoid is used so non-linearity can be applied to the range which leads to faster convergence. It uses back propagation which compute the partial derivatives of the cost function. It does this by using gradient decent to update the weights. This is was causes the actual learning.

If the data is not known then we will use the minimiser of the generalised cross validation function. This produces an e3xpected MSE which approaches the minimum possible expected MSE as $N \to \infty$This will only work if the data set is large enough to be able to distinguish X from X noise.

Lag by the system was caused by the calculation process but due to the step size being so small the robot had already moved on.

the hidden layer of an RBF is linear whereas the output later is not. However, in the MLP the hidden layers and the output layers are both non- linear, more over the RBF computes the Euclidean norm between input vector and centre of the unit. Whereas in the MLP computes the Synaptic weight vector of the unit. Furthermore, the XOR problem cannot be solved by the RBF. Lastly the MLP requires a smaller number of parameters than an RBF which leads to a higher degree of accuracy.

Conclusion

Parts one and two were making the robot seem as if it was moving and mimicking real life environment by adding noise which had to by tiny as the senses in a robot are often accurate.

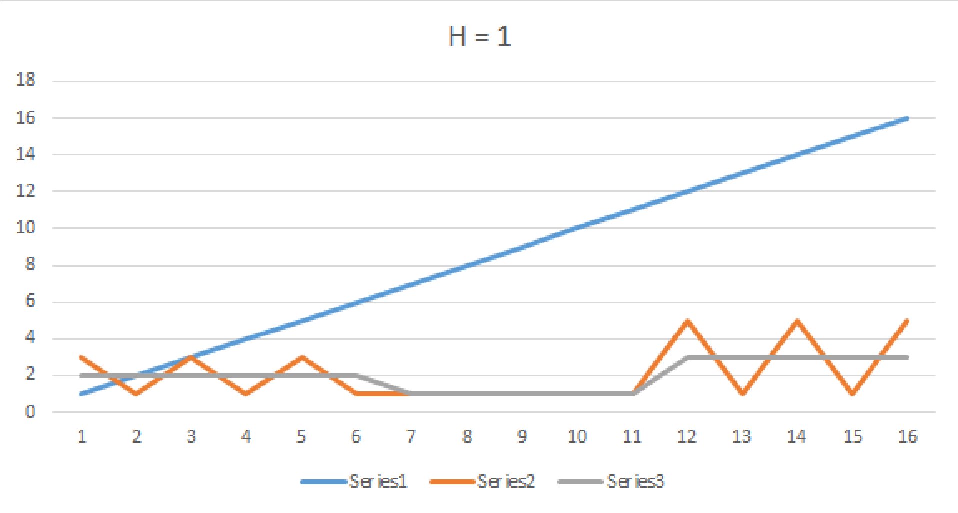
Also In this experiment we learnt that in order for the perception to be able to track the position of the robot effectively we needed to use a sigmoid, this allowed us to back propagate which is what caused the robot to learn.

528649 Ismail bhamjee

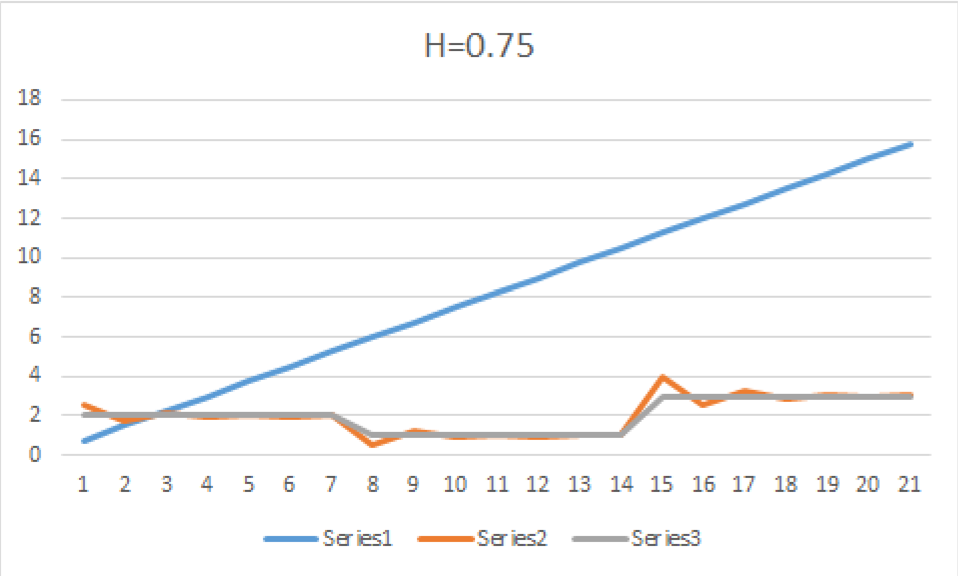
Code/ appendix

Part 1

H = 1

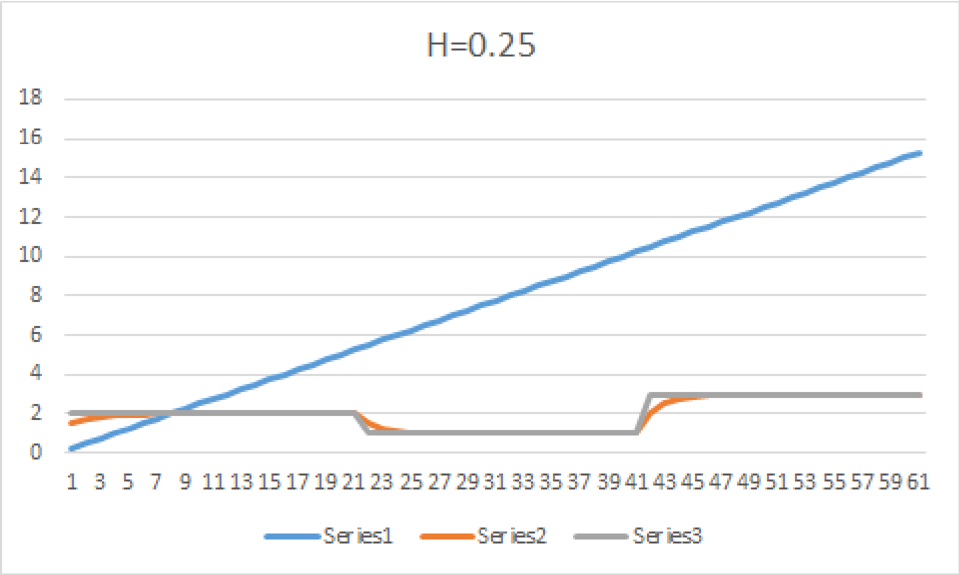


H = 0.75

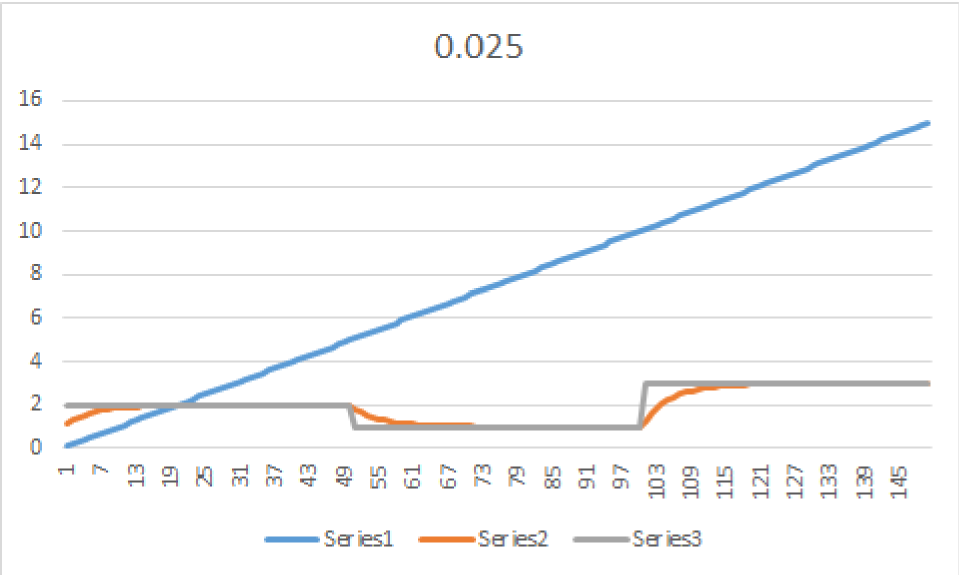


H = 0.25

528649 Ismail bhamjee

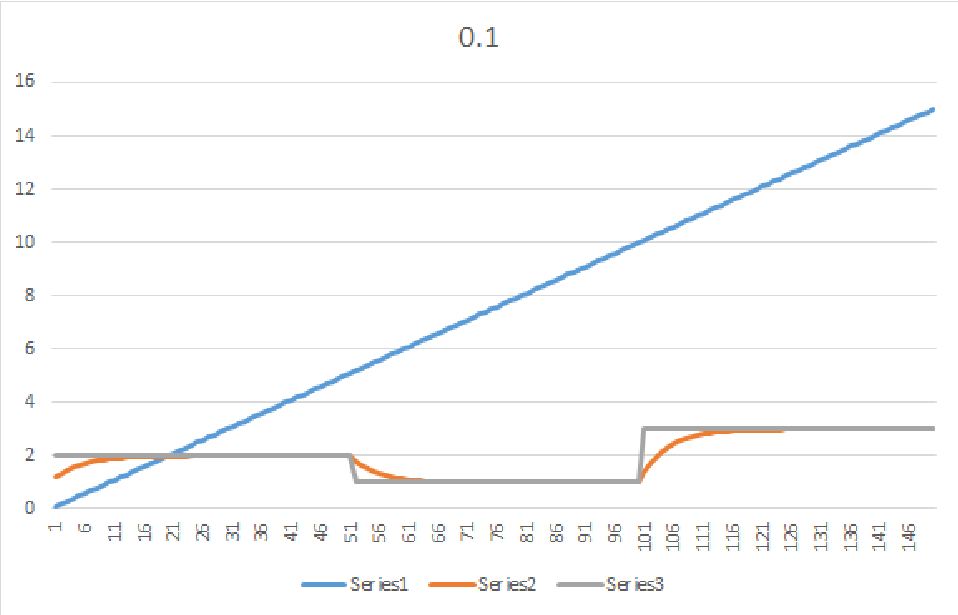


H= 0.025



H= 0.1

528649 Ismail bhamjee



Code for the variables

```
namespace perceptron
{
    class Program
    {


        static List<float> x = new List<float>();

        static float h = 0.001f;
        // time step // see lecture

        static float a = -2;
        static float b = 2;
        static float u = 0;


        static float tf = 15;

        // time at 0
        static float t = 0;


        static int k = 0;


        static void part1()...

        static void part2()
        {
```

Code for U change

528649 Ismail bhamjee

```
static void part1()
{
    Random r = new Random();

    r.NextDouble();
    double sample = 0.1 / h;
    int samplee = (int)sample;
    int counter = 0;
    using (StreamWriter write = new StreamWriter("data.csv"))
    {
        x.Add(1.0f);
        // adds 1.0f to the list

        while (t<= tf)
        {

            if (t <5)
            {
                u = 2;
            }
            else if (t<=10)
            {
                u = 1;

            }
            else if (t <=15)
            {
                u = 3;

            }

            x.Add(x[k] + h * (a * x[k] + b * u));

            // adds( 0 + step size + (-2 * x (k) + 2 * 0))
            t = t + h;
            k++;
            sample = samplee + 1;

            counter++;
            if (counter == samplee)
            {
                write.WriteLine(t + "," + x[k] + "," + u);

                counter = 0;
            }

        }
        write.WriteLine(t + "," + x[k] + "," + u);
    }
    double dist = r.NextDouble() * (2 * Math.PI);
}

static void part2()...
```

Part 2

Code

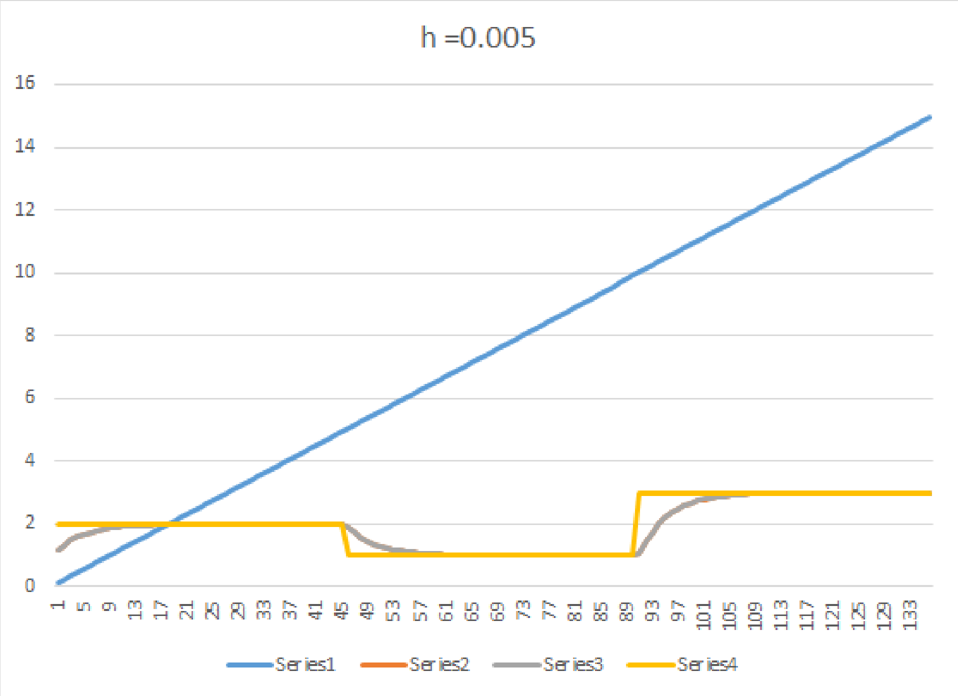
528649 Ismail bhamjee

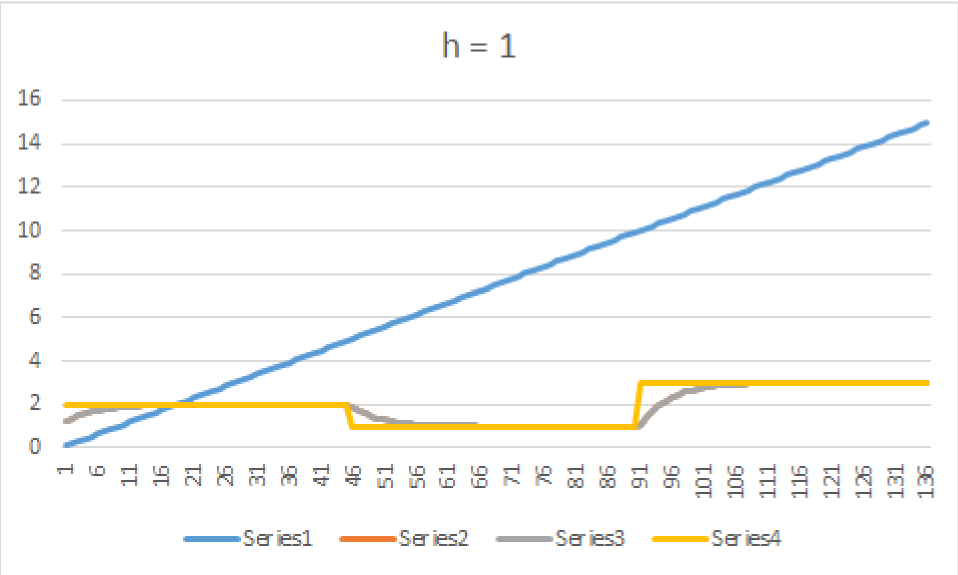
```
{


    int it = 0;
    double mean = 0.0;
    double SD = 0.001;
    double z2 = 0;

     for (int i = 0; i < part1.Count; i++)
    {

        string entry = part1[i];

        string[] entryarry = entry.Split(new char[] { ',' }, 3);


        string time = entryarry[0];
        string sstring = entryarry[1];
        // sstring is x from the old one

        double.TryParse(sstring, out double x);

        string u = entryarry[2];


        double a = r2.NextDouble() * (2 * Math.PI);

        double b = SD * Math.Sqrt(-2 * Math.Log(r2.NextDouble()));


        if(it == 0)

        {

            double z1 = b * Math.Sin(a) + mean;
            double xnoise = x + z1;
            writer.WriteLine(time + "," + x + "," + xnoise + "," + u);
            it = 0;
            z2 = b * Math.Cos(a) + mean;

        }

        else if (it == 1)
        {
            double xnoise = x + z2;
            writer.WriteLine(time + "," + x + "," + xnoise + "," + u);
            it = 0;

        }
    }




}
```

83 %

Output

528649 Ismail bhamjee





Part 3

528649 Ismail bhamjee

Program.cs

ConsoleApp1 ACW2.Program

```
string path = @"\\adir.hull.ac.uk\home\528\528649\Desktop\chnadra\working\ConsoleApp1\ConsoleApp1\bin\Debug\netcoreapp2.0\noisy.txt";

List<string> data = new List<string>();
using (StreamReader sr = File.OpenText(path))
{
    string line;
    while ((line = sr.ReadLine()) != null)
    {
        data.Add(line);
    }
}

List<float> input1 = new List<float>();


for (int i = 0; i < data.Count; i++)
{

    var text = data[i].Split(',').ToArray();
    string S4 = text[2];
    input1.Add(float.Parse(S4));

}

float max = input1.Max();

float min = input1.Min();

for (int i = 0; i < input1[i]; i++)
{
    input1[i] = ((input1[i] - min) / (max - min));
}


//declare RMSE as 5, do a while loop (while RMSE > 0.1), then do a nested for loop. the outer for loop goes through each sample. simulate the
//Each run through the while loop is 1 epoch, where it goes through the entire data set and trains. the outer for loop goes through each samp
for (int i = -2; i < input1.Count - 2; i++)
{
    Program p = new Program(3);
    float result;

    if (i == -2)
    {
        float[] point = { 0, 0, input1[0] };result = p.Feed(point);
```

528649 Ismail bhamjee

```
                if (i == -2)
                {
                    float[] point = { 0, 0, input1[0] };result = p.Feed(point);
                }
                else if (i == -1)
                {
                    float[] point = { 0, input1[0], input1[1] };result = p.Feed(point);
                }
                else
                {
                    float[] point = { input1[i], input1[i + 1], input1[i + 2] };result = p.Feed(point);
                }

                Console.WriteLine(result);
            }

        }

        Program(int n)
        {
            weights = new float[n];

            for (int i = 0; i < weights.Length; i++)...
        }

        float Feed(float[] inputs)
        {


            float sum = 0;
            for (int i = 0; i < weights.Length; i++)
            {
                sum += inputs[i] * weights[i];
            }
            return Activate(sum);
        }

        float Activate(float sum)
        {
            if (sum > 0) return 1;
            else return 0;
        }
```

```
        1 reference
        float Activate(float sum)
        {
            if (sum > 0) return 1;
            else return 0;
        }
        0 references
        float Sigmoid(float sum)
        {
            return (float)(1.0 / (1.0 + Math.Pow(Math.E, -sum)));
        }
        0 references
        void Train(float[] inputs, int desired)
        {
            float guess = Feed(inputs);

            float error = desired - guess;
            for (int i = 0; i < weights.Length; i++)
            {
                weights[i] += c * error * inputs[i];
            }
        }


    }
}
```