Simulation and intelligent tracking of a robot

Report

Submitted for:
600093 Computational Science

4th April 2019

by

Ieaun Roberts

Word Count:
1453

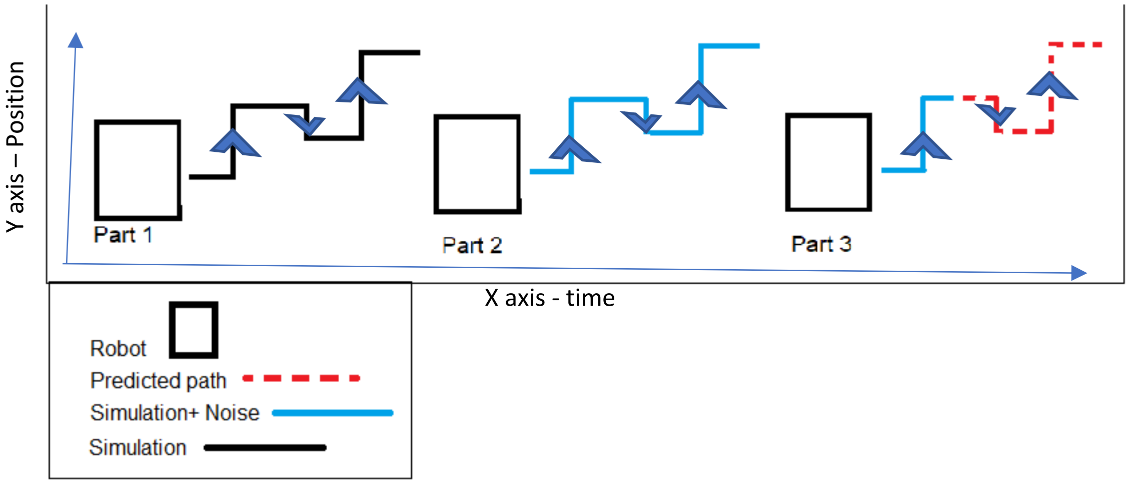
Table of contents

Introduction

Within this report documents the findings and experimental results obtained while completing the coursework for the module "Computational Science". The project touches on several topics such as simulation of a robot in 1 dimension, tracking said robot and applying noise to the obtained results as would be the cause in a real-world context and finally leading into more complex systems such as perceptron's and machine learning to predict the robot's movements.

Breakdown of the project:

- Part 1 : Simulate robot
- Part 2 : Add Noise to Simulated results
- Part 3 : Developed an intelligent agent to predict the next position of the robot



[Figure 1, Breakdown of the Acw, movement indicated by arrows]

Part 1 : Simulation

In this section we are given a simplified model of the robot:

$$\dot{x} = -2x + 2U$$

As time changes, so does our movement value U .This value represents the distance from the origin the robot must travel, we begin at an origin of 0.

$$U = \begin{cases} 2\ for\ 0 < t \leq 5 \\ 1\ for\ 5 < t \leq 10 \\ 3\ for\ 10 < t \leq 15 \end{cases}$$

H =Integration step size
T = Time (Seconds)
U = Distance from origin to travel
X= generalized co-ordinates as a distance from the origin
K = sample number

Simulation Algorithm outline

```
Initialize Variables (K,h,e,a,samples) and Lists (X,Time,U,FileNames)
        XValues, TimeValues, UValues = hValues(StepSize)
                Create/open file in write (str(StepSize)+"txt")
                Step2(K,h,e,a,samples X,Time,U,FileNames)
                        while time[k] <= 15
                            if time[k] <= 5                     #Change U according to time
                              U = 2
                            elif 5 < time[k] <= 10
                              U = 1
                            elif 10 < time[k] <= 15
                              U = 3
                            X.append(x[k] + h * (a*x[k]+(2*U))) # Holds all X values
                            U.append(U)                        #Holds all changes in U
                            Time.append(Time[k] + h)            #Holds all time values
                            K++                                 #counter
                            Samples++
                            sampleInterval = h*(number of steps)
                              if sampleInterval == 10:
                                if samples == 10:               #tenth step if h=0.01
                                  write data to file()
                                  samples = 0
```

[Figure 2 represents Simulation process]

[Figure 3 right , time = 0.01] Explores the forward difference Euler's Method which is used to calculate the next value of X(k) which contains the current position of the robot. With this algorithm in place it can be used to find the value of X(t) at any given time.

H = 0.01

X(t0) = 0 / X(0) = 0 (origin/starting point = 0)

Next point = current point + step * model

X (0.01) = X(0) + 0.01 * (-2x + 2U)

=0 + 0.01 * (-2 * (0) + 2U)

if 0 < t <= 5 ------→ **U= 2**

if 5 < t <= 10 ------→ **U= 1**

if 10 < t <= 15 ------→ **U= 3**

=0 + 0.01*(0 + 2(2))

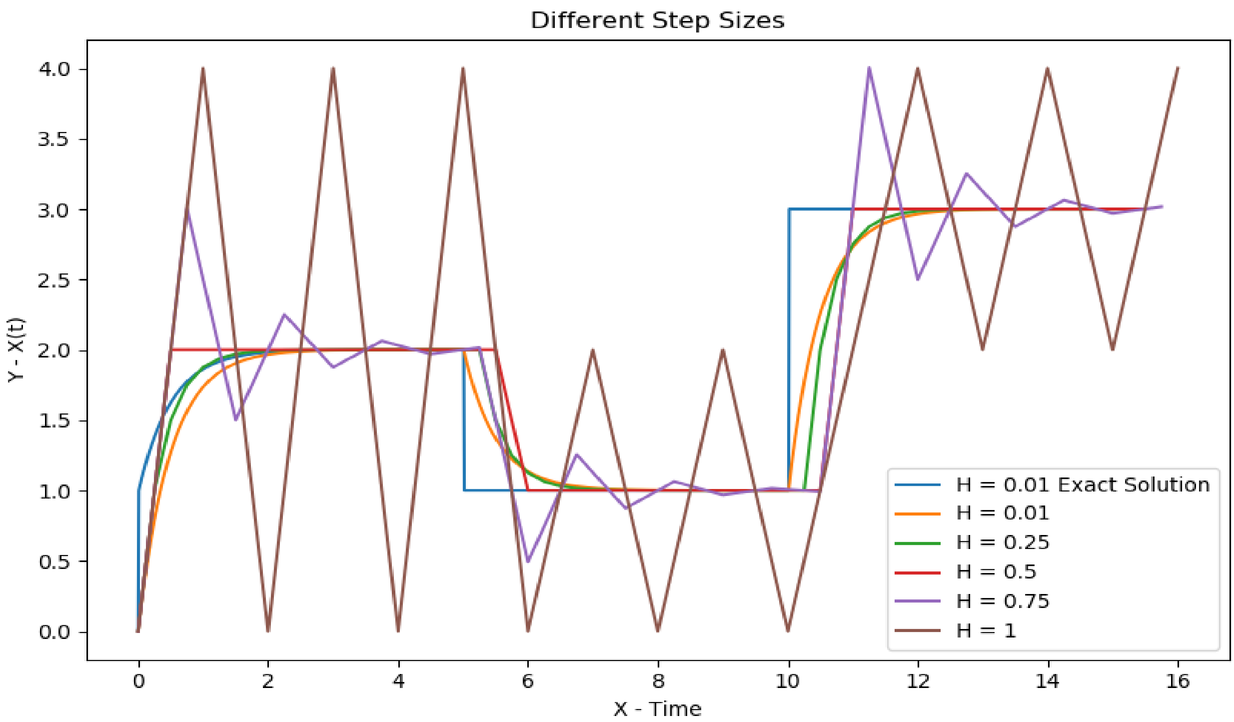
=0 + 0.01*(4)

= 0.04

Exact solution:

$$x(t) = U(t) - e^{-2t}$$

From this [Figure bellow] we can deduce that reasonably smaller values of h provide more accurate results (h < 1). As the value of H approaches 0 the error between the exact solution and the simplified solution becomes smaller. Using the same logic, higher values of h therefore increase the error causing the simulation to become unstable. **Smaller step size = better approximation** and larger step sizes cause overshoots which attempt to recover over time (as with H = 1 a continual overshoot loop is entered/ with H = 0.75 an overshoot is recorded and then it attempts to recover reducing the error as the robot remains stationary).



[Figure 4 comparison of different step sizes with exact solution]

Upper Limit ?

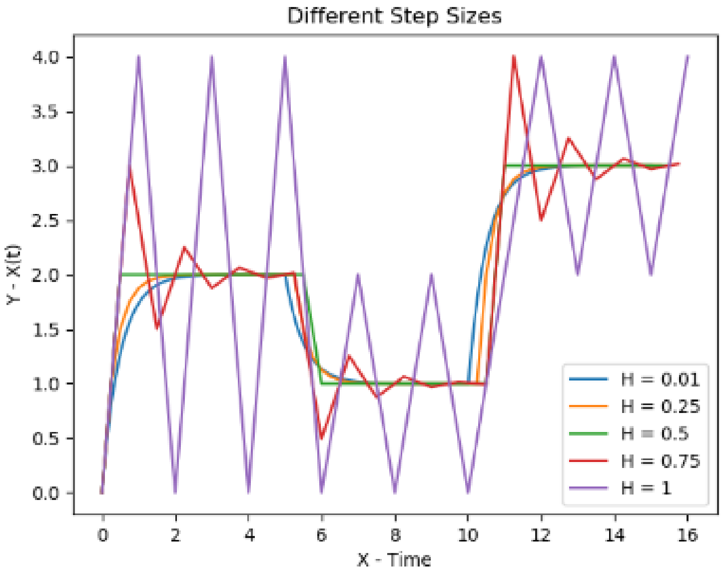
$\dot{x} = ax; and\ h\ is\ step\ size. The\ numerical\ simulation\ is\ unstable\ if$

$|ha + 1| \leq 1$

The upper value of h for the model is 0.5, it is the highest value that provides the least overshoot/ need for recovery, is computationally efficient and after following the method outlined within the lectures, the product of h = 0.5 and a = -2 is 0 which is not negative and smaller than 1. Any step size above this value becomes computationally unstable which is evident in the error accumulation driven overshoots shown by H=1 and H = 0.75 [Figure below]

H Values	Method
0.01	0.01(-2) + 1 = 0.98 **0.98 <= 1** **stable**
0.25	0.25(-2) + 1 = 0.5 **0.5 <= 1** **stable**
0.5	0.5(-2) +1 = 0 **0 <= 1** **stable**
0.75	0.75(-2) + 1 = -0.5 **-0.5 <= 1** **Unstable**
1	1(-2) + 1 = -1 **-1 <= 1** **Unstable**

If the product of |ha +1| <= 1 then the simulation is unstable:



The lower limit is H = 0.01 as any value below this results in practically identical plots but takes longer and more power to compute.

Part 2 : Box Muller Noise

The objective of Part 2 was to add noise using the Box-Muller method in a normal distribution to our results from Part 1 with a Standard Deviation = 0.001 and a Mean = 0.0.

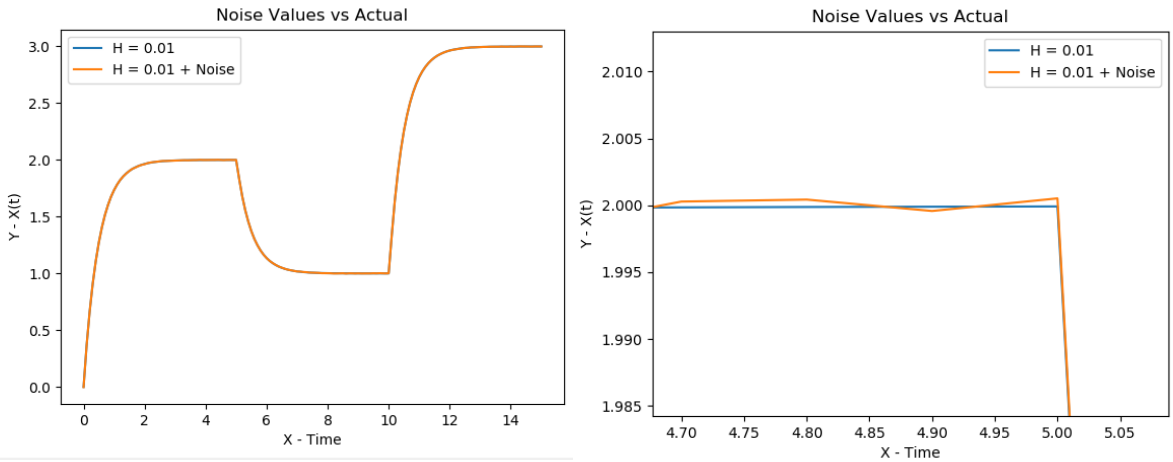
The Box- Muller method generates 1 random number in a uniform distribution and returns two random numbers normally distributed. These random numbers help to add noise (white noise) to our values to simulate that of real-world sensor systems.

The box muller method can be broken down into 3 steps:

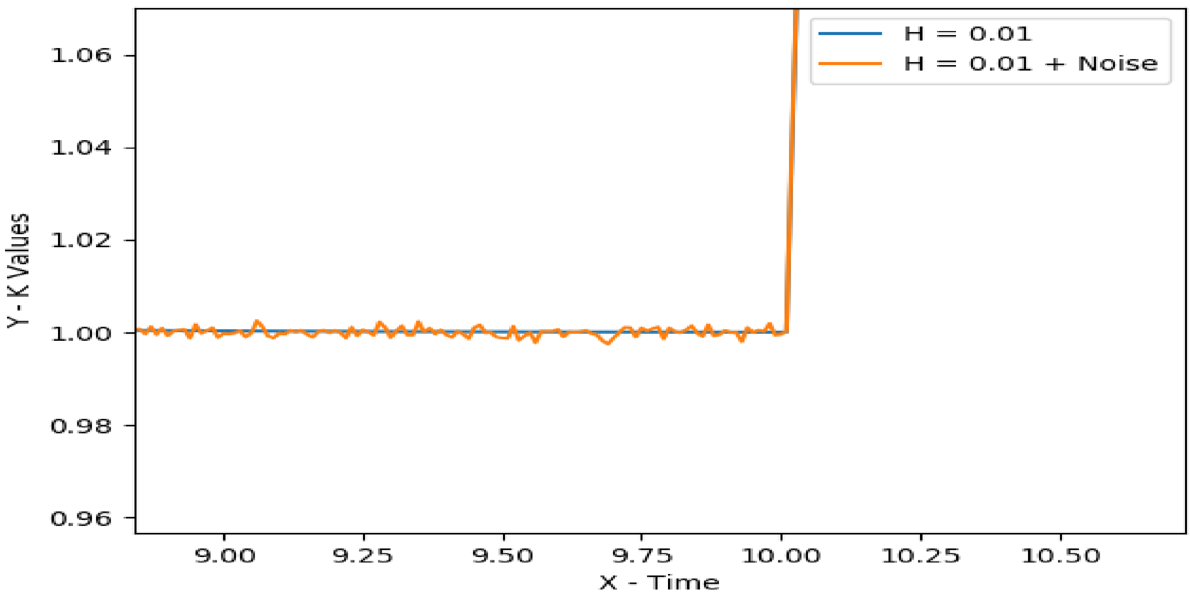
- Step 1 – Generate a uniformly distributed random number a in [0,2* π]
- Step 2 - Compute $B = \sigma\ sqrt(\ -2\ *\ natural\ log\ (\ random\ number\ (0,1)\ )\ )$
- Step 3 - Compute $X1 = b\ sin(a) + \mu$ and $X2 = bcos(a)+\mu$

These numbers generated by the box muller method are normally distributed ,because items in this distribution are all based around a mean and standard deviation value so that they accurately represent the data. Normal distribution values tend to cluster closer around an average result vs complete randomness that has 1 in max probability that is used in Uniformly distributed methods.

The Box muller method receives one uniformly distributed random number and generates two normally distributed random numbers, it does this by substituting b into both equations shown in step 3 (Cos and Sin functions) and then returns one and saves the other for the next request for a random number.



[Figure 5 represents Noise values vs actual, note {Right} demonstrates the deviation of the noise results]



Part 3

In part 3 an intelligent agent in the form of a perceptron was created to predict the next position of the robot. The program makes use of the added noise positional information from Part 2 to train the Perceptron.

Perceptron weight training can be broken down into 3 steps:

1. **Calculate Sum**: Input values multiplied by respective weights to get weighted sum
    a. {W0 + X1W1 + X2W2+ XnWn}
2. **Calculate Output**: This expression is added together inside the agent and enters an activation function which decides if the sum is over a certain threshold on outputting either 0 or 1.
3. **Update Weights**: This output is then subtracted from the actual positional value to get an error

This error added to the target value gives us our perceptron's expected value.

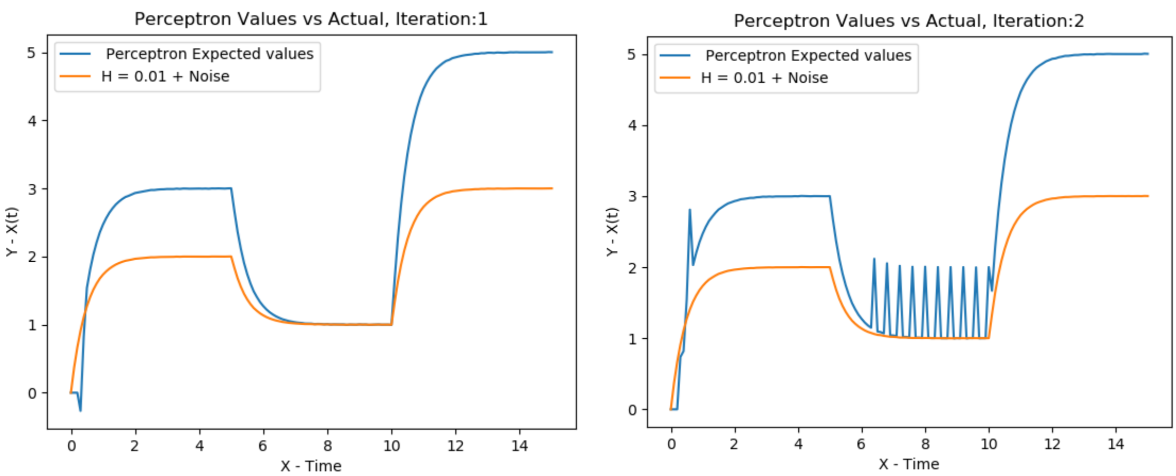
The perceptron is initialized with 3 random weights and a random threshold all within the range (-0.5, 0.5) a bias of 0.5 and a learning rate of 0.2

The figure below represents the algorithm used to calculate the error:

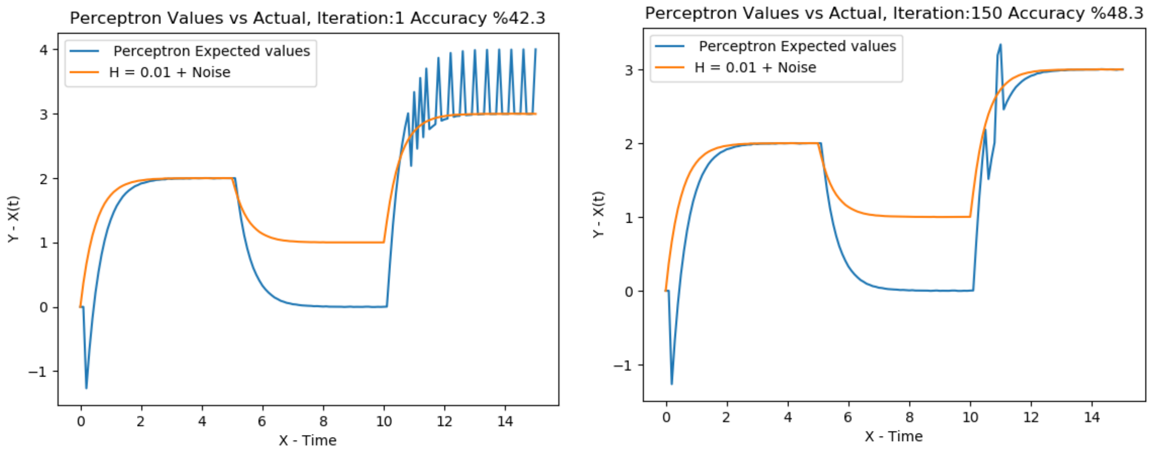
Iteration	Input (1)	Input (2)	Input (3)	Output of perceptron	error	Error Result
x	X(1)	X(2)	X(3)	X_p(4)	X(4) - X_p(4)	
x	X(2)	X(3)	X(4)	X_p(5)	X(5) - X_p(5)	
1	0	0	0.000542	0	0.000542 – 0	0.000542
2	0	0.000542	0.365854	0	0.664784 – 0	0.664784
3	0.000542	0.365854	0.664784	0	0.909031 – 0	0.909031
4	0.365854	0.664784	0.909031	1	1.1085991 – 1	0.1085991
5	0.664784	0.909031	1.1085991	1	1.2716606 – 1	0.2716606

From this figure we can deduce that when the perceptron outputs a 0 our weight gain increases much faster in proportion to the last increase. An output of 1 from the perceptron decreases this gain and also cause negative gain.

Initially the perceptron is incapabale of learning the data with its step activation function as it is not linearly separable, it cannot differentiate due to the curve created by each new U step (3,1,2). It can predict the rate of change for the next position after training against its trajectory but as soon as a step is introduced it is unable to predict the next position and must begin adjusting its weights again for the new trajectory. Step Functions result in 1 of 2 decisions, so the one of the 3 values of U cannot be accounted for. A single neuron doesn't work for continous variables.

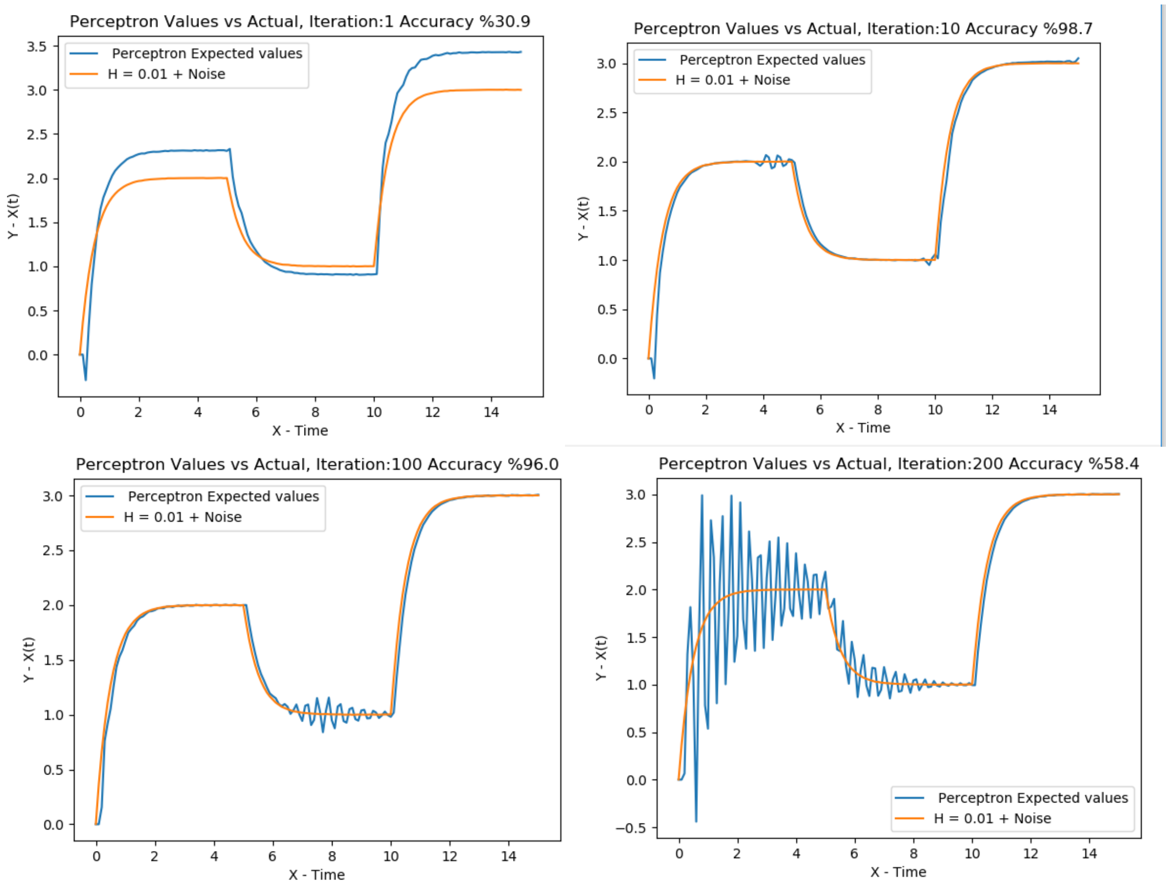


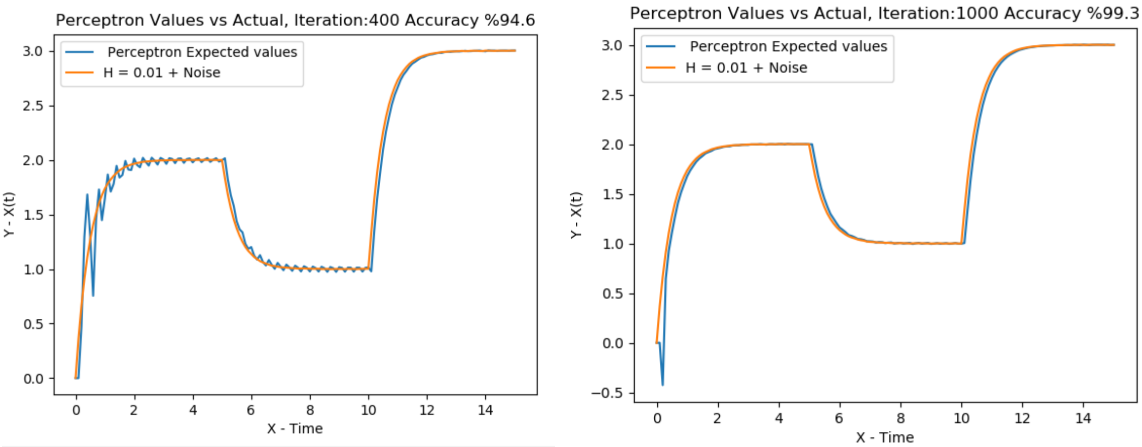
[Figure 6 representation of the step activation function returning either 1 or 0]



[Figure 7 representation of the step activation function returning either 2 or 3]

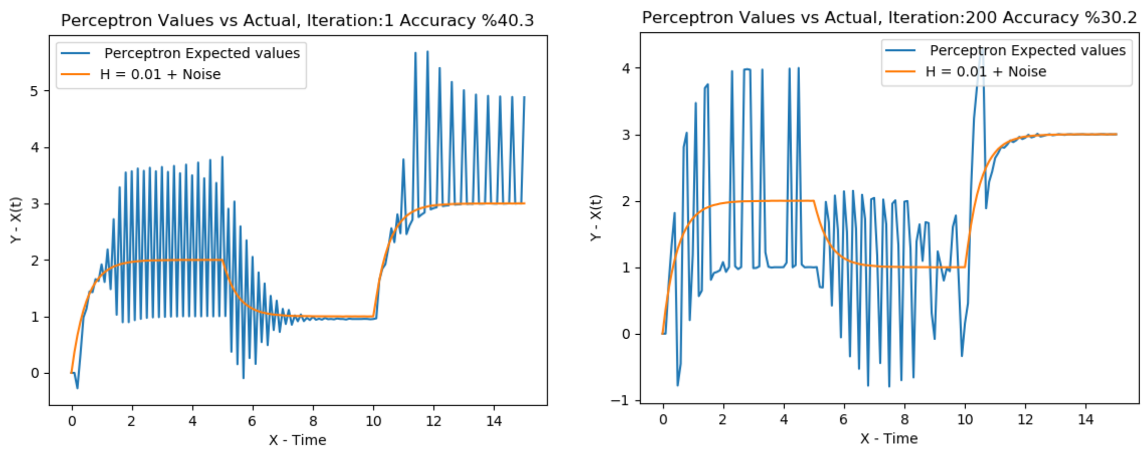
After replacing the activation function with a logistic sigmoid the following results were obtained:





[Figure 8 lifecycle of Perceptron predicted values vs actual values at a learning rate of 0.2]

It takes some time to compute since for each iteration the perceptron must go over each input and adjust each weight again individually ,but it is evident that the neuron is now capable of learning. After each iteration the output of the perceptron is calculated by the logistic sigmoid which results in the weights being adjusted accordingly in an exponential manner rather than binary. After many epochs the error is driven down to a difference of 0.0002 in 99.3% of the plots of actual data vs the predicted. However, when making large changes to the learning rate such as an increase from 0.2 to 0.9 it is unable to adapt



[Figure 8 representing an altered learning rate of 0.9]

As we have already introduced noise into our results from Part 1 and the Agent is able to reduce the error to near 0 with said results it is a nod towards the agent being able to generalize.

The perceptron could be extended to predict both position and velocity by making use of the timing array already implemented and the ***kinematics equation of motion***: S=VT where S = distance V = Velocity and T = Time. 1st the perceptron would need to predict the location of the robot which is already functional, then it would just be a simple matter of (within the context of the already created software):

- k = 0 = (0.1 seconds as every 10th 0.01 H)
- TimeStart = time[k]
- TimeEnd = time[k+9]
- Time = TimeEnd – TimeStart
- DistanceStart = ExpectedValues[k] #Position (calculated by perceptron)
- DistanceEnd = ExpectedValues[k+9]
- Distance = DistanceEnd - DistanceStart
- Velocity = Distance / Time

Or Velocity = ((ExpectedValues[k+10] -ExpectedValues[k]) / (time[k+10] - time[k])

Conclusion

Step Sizes of H are crucial, anything over the upper limit causes overshoots and instability and anything under the lower limit leads to computational inefficiency.

Adding noise using the methods outlined can cause simulation systems to appear identical to their real-world counterparts especially in regard to noise encountered when using sensors, learning how to use these functions will be a valuable asset.

Numerical systems in tandem with logical functions can be made use of to adjust values over time and generate errors, retention of this information and accuracy of executions to combat these errors can result in the creation of learning algorithms and neural networks that can react to data in ways that may seem meaningful based on the situation. As evident in the later part of the coursework limitations are bound to arise in these sorts of systems such as the binary output of the activation function which was replaced with a sigmoid. Finding a way around limitations and adapting to using a model-based approach have been two key skills that I feel I have learnt while completing this coursework.

Appendix: Text Files 0.01 + noise

```
Time =0 X =0 Xnoise =0.00018901644806726774 U=0
Time =0.09999999999999999 X =0.3658543862249063 Xnoise =0.36631018184625863 U=2
Time =0.20000000000000004 X =0.664784056489811 Xnoise =0.6651052035453403 U=2
Time =0.30000000000000001 X =0.9090313612351256 Xnoise =0.9083912743473146 U=2
Time =0.40000000000000002 X =1.108599192098098 Xnoise =1.1096553541353231 U=2
Time =0.50000000000000002 X =1.2716606398257657 Xnoise =1.27325283232967 U=2
Time =0.60000000000000003 X =1.4048937146157585 Xnoise =1.403972483044274 U=2
Time =0.70000000000000004 X =1.5137548370046763 Xnoise =1.5149390647399639 U=2
Time =0.80000000000000005 X =1.6027022998359184 Xnoise =1.6018549683637093 U=2
Time =0.90000000000000006 X =1.6753788529569669 Xnoise =1.6750506269569325 U=2
Time =1.00000000000000007 X =1.734760888210494 Xnoise =1.735661187130808 U=2
Time =1.10000000000000008 X =1.7832803344337884 Xnoise =1.782730050936181 U=2
Time =1.20000000000000008 X =1.8229242545480853 Xnoise =1.82246779619916 U=2
Time =1.3000000000000001 X =1.8553162236318992 Xnoise =1.8544239128369342 U=2
Time =1.4000000000000001 X =1.881782820731826 Xnoise =1.8825997544560007 U=2
Time =1.5000000000000001 X =1.903407957513025 Xnoise =1.9024942965678786 U=2
Time =1.60000000000000012 X =1.921077268722166 Xnoise =1.9210182183985982 U=2
Time =1.70000000000000013 X =1.9355143824275887 Xnoise =1.9357035673491567 U=2
Time =1.80000000000000014 X =1.9473105554462329 Xnoise =1.9449853473877552 U=2
Time =1.90000000000000015 X =1.9569488876451078 Xnoise =1.9570542312782182 U=2
Time =2.00000000000000013 X =1.9648241067885572 Xnoise =1.9648798066371889 U=2
Time =2.0999999999999999 X =1.97125873419895 Xnoise =1.9710299642453264 U=2
Time =2.1999999999999997 X =1.976516293278435 Xnoise =1.9764795826544952 U=2
Time =2.2999999999999995 X =1.980812101832887 Xnoise =1.9816698051428012 U=2
Time =2.3999999999999993 X =1.9843220901863245 Xnoise =1.9855438287756026 U=2
Time =2.49999999999999907 X =1.9871900062224104 Xnoise =1.9878667287400977 U=2
Time =2.59999999999999885 X =1.9895333024279327 Xnoise =1.9902048720157248 U=2
Time =2.69999999999999864 X =1.9914479460359478 Xnoise =1.9921643535525502 U=2
Time =2.79999999999999843 X =1.993012349262938 Xnoise =1.9919706756191309 U=2
Time =2.8999999999999982 X =1.9942905805987188 Xnoise =1.9932502928922822 U=2
Time =2.99999999999998 X =1.9953349886640968 Xnoise =1.9945314729909038 U=2
Time =3.0999999999999978 X =1.9961883460936112 Xnoise =1.9960227345380601 U=2
Time =3.19999999999999758 X =1.996885601243823 Xnoise =1.9972046834429007 U=2
Time =3.29999999999999736 X =1.9974553094665235 Xnoise =1.9987590302622298 U=2
Time =3.39999999999999715 X =1.997920802563152 Xnoise =1.9981902605524293 U=2
Time =3.49999999999999694 X =1.9983011443142014 Xnoise =1.9972629735268628 U=2
Time =3.59999999999999672 X =1.9986119112163077 Xnoise =1.9998447588846135 U=2
Time =3.699999999999965 X =1.9988658304012996 Xnoise =1.9993893059848549 U=2
Time =3.799999999999963 X =1.9990733008625035 Xnoise =1.9996063920048757 U=2
Time =3.899999999999961 X =1.9992428193345853 Xnoise =2.0003753341107844 U=2
Time =3.99999999999999587 X =1.9993813282683885 Xnoise =1.9993528586176976 U=2
Time =4.099999999999957 X =1.9994945001517104 Xnoise =1.9987684607752387 U=2
Time =4.199999999999955 X =1.9995869698200772 Xnoise =1.9993009616269297 U=2
Time =4.299999999999953 X =1.9996625242715613 Xnoise =2.0014220608293423 U=2
Time =4.399999999999951 X =1.9997242577593084 Xnoise =2.0013596575685235 U=2
Time =4.49999999999999485 X =1.9997746985134206 Xnoise =2.000273468792018 U=2
Time =4.599999999999946 X =1.9998159122819648 Xnoise =1.9990985854284953 U=2
Time =4.699999999999944 X =1.9998495869315114 Xnoise =2.000454742551862 U=2
Time =4.799999999999942 X =1.9998771015719374 Xnoise =2.0004293530375485 U=2
Time =4.89999999999994 X =1.999899583036421 Xnoise =1.9986553878243898 U=2
Time =4.999999999999938 X =1.9999179520297095 Xnoise =2.000742353168383 U=2
Time =5.099999999999936 X =1.8336807229647651 Xnoise =1.8321996527107003 U=1
Time =5.199999999999934 X =1.68117784836086 Xnoise =1.6808411919163586 U=1
Time =5.299999999999931 X =1.5565718965498276 Xnoise =1.5552565253054924 U=1
Time =5.399999999999929 X =1.4547597617486931 Xnoise =1.4558009934942657 U=1
Time =5.499999999999927 X =1.371571834991517 Xnoise =1.3725420716302903 U=1
Time =5.599999999999925 X =1.303601242176875 Xnoise =1.3020576705036442 U=1
Time =5.699999999999923 X =1.2480643191200054 Xnoise =1.247609918785271 U=1
Time =5.799999999999921 X =1.2026866095120312 Xnoise =1.2041704798439743 U=1
Time =5.899999999999919 X =1.1656097169525155 Xnoise =1.1665592557300994 U=1
Time =5.99999999999999165 X =1.135315196278244 Xnoise =1.1353124325545907 U=1
Time =6.099999999999914 X =1.1105623672376042 Xnoise =1.1106181751614241 U=1
Time =6.199999999999912 X =1.0903375037349612 Xnoise =1.0896808354616296 U=1
```

```
Time =8.699999999999859 X =1.0005786114303634 Xnoise =1.00125860615816 U=1
Time =8.799999999999857 X =1.000472767665504 Xnoise =1.0007012101719994 U=1
Time =8.899999999999855 X =1.0003862856034593 Xnoise =1.0002291197523039 U=1
Time =8.999999999999853 X =1.0003156234622788 Xnoise =0.9976710589684028 U=1
Time =9.09999999999985 X =1.0002578873482437 Xnoise =1.000884801695717 U=1
Time =9.199999999999848 X =1.0002107127394901 Xnoise =0.9992074005382019 U=1
Time =9.299999999999846 X =1.0001721676495021 Xnoise =0.999373561751094 U=1
Time =9.399999999999844 X =1.000140673504634 Xnoise =1.0007194957812686 U=1
Time =9.499999999999842 X =1.0001149404952858 Xnoise =0.9991852166444107 U=1
Time =9.59999999999984 X =1.000093914753108 Xnoise =1.000515147518777 U=1
Time =9.699999999999838 X =1.0000767351909299 Xnoise =1.0004633844268647 U=1
Time =9.799999999999836 X =1.00006269823784 Xnoise =0.9985710508703972 U=1
Time =9.899999999999833 X =1.000051229025179 Xnoise =0.9994338169477709 U=1
Time =9.999999999999831 X =1.0000418578433972 Xnoise =1.0002906303562402 U=1
Time =10.09999999999983 X =1.332538676645295 Xnoise =1.332744055260362 U=3
Time =10.199999999999827 X =1.6375626961501473 Xnoise =1.6378577062087263 U=3
Time =10.299999999999825 X =1.8867895279350992 Xnoise =1.887750568004718 U=3
Time =10.399999999999823 X =2.0904259949333204 Xnoise =2.090370875470419 U=3
Time =10.49999999999982 X =2.256811814608221 Xnoise =2.2566073565259614 U=3
Time =10.599999999999818 X =2.3927611433162768 Xnoise =2.3924008286178444 U=3
Time =10.699999999999816 X =2.5038416429182453 Xnoise =2.5025111543593295 U=3
Time =10.799999999999814 X =2.5946024985184963 Xnoise =2.5943558964878504 U=3
Time =10.899999999999812 X =2.6687607255593098 Xnoise =2.6675476336722688 U=3
Time =10.99999999999981 X =2.7293533962813505 Xnoise =2.7274487733974824 U=3
Time =11.099999999999808 X =2.778862019825022 Xnoise =2.7798409180508417 U=3
Time =11.199999999999806 X =2.8193141698289876 Xnoise =2.818940143500282 U=3
Time =11.299999999999804 X =2.8523665215773644 Xnoise =2.8533521474514525 U=3
Time =11.399999999999801 X =2.8793726993946454 Xnoise =2.8795371054758254 U=3
Time =11.4999999999998 X =2.9014387129071153 Xnoise =2.901873620937146 U=3
Time =11.599999999999797 X =2.9194682525045677 Xnoise =2.9198674314942723 U=3
Time =11.699999999999795 X =2.9341996990303483 Xnoise =2.9339621114646284 U=3
Time =11.799999999999793 X =2.9462363633926816 Xnoise =2.947139849584301 U=3
Time =11.89999999999979 X =2.9560711945287768 Xnoise =2.9585163995913413 U=3
Time =11.999999999999789 X =2.96410696761041 Xnoise =2.9660201762065 4 U=3
Time =12.099999999999786 X =2.9706727792777325 Xnoise =2.970670085828244 U=3
Time =12.199999999999784 X =2.976037525446247 Xnoise =2.9762625751336844 U=3
Time =12.299999999999782 X =2.9804209136563933 Xnoise =2.981164623826306 U=3
Time =12.39999999999978 X =2.984002460964935 4 Xnoise =2.982899320959577 U=3
Time =12.499999999999778 X =2.986928845877327 Xnoise =2.9861287457465027 U=3
Time =12.599999999999776 X =2.989319915411728 Xnoise =2.9889165271286 6 U=3
Time =12.699999999999774 X =2.991273593307664 Xnoise =2.991523028082446 U=3
Time =12.799999999999772 X =2.9928698903898505 Xnoise =2.9926774968570333 U=3
Time =12.89999999999977 X =2.994174181327419 Xnoise =2.994809945136453 U=3
Time =12.999999999999767 X =2.9952398819847765 Xnoise =2.995258291635403 U=3
Time =13.099999999999765 X =2.9961106370121855 Xnoise =2.997624754675478 U=3
Time =13.199999999999763 X =2.9968221072665413 Xnoise =2.9956322052603372 U=3
Time =13.299999999999761 X =2.9974034302642854 Xnoise =2.997922765489975 U=3
Time =13.399999999999759 X =2.99787841347776 Xnoise =2.9976341407205376 U=3
Time =13.499999999999757 X =2.9982665093452185 Xnoise =2.9973154618052615 U=3
Time =13.599999999999755 X =2.998583611924984 Xnoise =2.998468919204178 U=3
Time =13.699999999999752 X =2.998842707819905 Xnoise =3.000387809187286 U=3
Time =13.79999999999975 X =2.9990544080300205 Xnoise =2.9998466281309883 U=3
Time =13.899999999999748 X =2.999227382514919 Xnoise =2.9980287995489467 U=3
Time =13.999999999999746 X =2.999368715262815 Xnoise =2.997329540048301 U=3
Time =14.099999999999744 X =2.999484194407843 Xnoise =2.9998615305372094 U=3
Time =14.199999999999742 X =2.999578549277009 Xnoise =2.9977819869541444 U=3
Time =14.29999999999974 X =2.999655644074801 Xnoise =2.9984067798555953 U=3
Time =14.399999999999737 X =2.999718636137629 Xnoise =3.000930956231329 U=3
Time =14.499999999999735 X =2.9997701052392154 Xnoise =3.0011427532272217 U=3
Time =14.599999999999733 X =2.999812159242517 Xnoise =2.997918461586485 U=3
Time =14.699999999999731 X =2.999846520425035 Xnoise =3.00056831713884 U=3
Time =14.799999999999729 X =2.9998745960128836 Xnoise =2.999136058331157 U=3
Time =14.899999999999727 X =2.9998975358122517 Xnoise =2.9994957301550524 U=3
Time =14.999999999999725 X =2.9999162792985112 Xnoise =3.0012293294099717 U=3
```

Appendix Code:

```
import numpy as np
import matplotlib.pyplot as plt
import random
import math
from random import randrange, uniform

# Open a new file/ clear the previous file with same name
def startfile(filename):
    LogFile = open(filename, "w")
    LogFile.write("")
    return


# append each new line of data to the file
def writefile(Data,filename):
    LogFile = open(filename, "a")  # r = read, w = write , a=append (write
    LogFile.write(Data)
    LogFile.close()
    return


# read the data from a file
def readfile(name):
    file = open(name, "r")
    return file


def step1(Step):
    k = 0
    time = []
    time.append(0) # Time
    h = Step  # Select h
    e = 1  # e>0
    x = []
    a= -2
    x.append(0)     # x[k] = 0
    samples = 0
    return k,a, time, h, e, x, samples
```

```python
def step2(k,a, time, h, e, x, u, samples,insertfilename):
    # dx/dt = x`= ax + 2U
    # =-2x + 2U
    while time[k] <= 15:
        ################ Step 2a
        if time[k] <= 5:
            U = 2
        elif 5 < time[k] <= 10:
            U = 1
        elif 10 < time[k] <= 15:
            U = 3
        ################# Step 2b    x[k+1] = x[k] + h * (a*x[k]+(2*U))
        x.append(x[k] + h * (a*x[k]+(2*U)))
        u.append(U)
        ################ Step 2c (array Variable time holds the specific time at k, go to next time a
        time.append(time[k] + h)
        k += 1
        ############# Step 2d
        samples += 1
        ###############Step 2e
        if h >= 0 and h <= 0.1:
            #sampleInterval = h*(number of steps)
            sampleInterval = round((0.1/h),2)
            if sampleInterval == 10:
                if samples == 10: #tenth step if h=0.01
                    writefile("\n" + " X(k):" + str(round(x[k], 4)) + " U(k):"
                              + str(round(U, 4)) + " Time:" + str(round(time[k],2)),insertfilename)
                    samples = 0
    return x,time,u
```

```python
def ExactSolution(k,a, time, h, e, x, u, samples,insertfilename):
    while time[k] <= 15:
        ################ Step 2a
        if time[k] <= 5:
            U = 2
        elif 5 < time[k] <= 10:
            U = 1
        elif 10 < time[k] <= 15:
            U = 3
        ################# Step 2b    x[k+1] = x[k] + h * (a*x[k]+(2*U))
        #x.append(x[k] + h * (a * x[k] + (2 * U)))

        ExactSo = U - math.exp((-2 * time[k]))
        x.append(ExactSo)
        u.append(U)
        ################ Step 2c (array Variable time holds the specific time at k, go to next time and counter
        time.append(time[k] + h)
        k += 1
        ############# Step 2d
        samples += 1
        ###############Step 2e
        if h >= 0 and h <= 0.1:
            # sampleInterval = h*(number of steps)
            sampleInterval = round((0.1 / h), 2)
            if sampleInterval == 10:
                if samples == 10:  # tenth step if h=0.01
                    writefile("\n" + " X(k):" + str(round(x[k], 4)) + " U(k):" + str(round(U, 4))
                              + " Time:" + str(round(time[k], 2)), insertfilename)
                    samples = 0
    return u
```

```python
    # Plot the axis of the recoded data
def PlotAxis():
    # plt.subplot(1, 3, 1)
    plt.plot(ArrayTime5, ArrayX5, label="H = 0.01 Exact Solution")
    plt.plot(ArrayTime0,ArrayX0,label= "H = 0.01")
    plt.plot(ArrayTime1,ArrayX1, label= "H = 0.25")
    plt.plot(ArrayTime2, ArrayX2, label= "H = 0.5")
    plt.plot(ArrayTime3, ArrayX3, label= "H = 0.75")
    plt.plot(ArrayTime4, ArrayX4, label= "H = 1")
    plt.xlabel('X - Time')
    plt.ylabel('Y - X(t)')
    plt.title("Different Step Sizes")
    plt.legend(loc='best')
    plt.show()
    return


def plotNoise(NoiseArray,ArrayX0 ,ArrayTime0):
    plt.plot(np.array(ArrayTime0), np.array(ArrayX0), label="H = 0.01")
    plt.plot(np.array(ArrayTime0), np.array(NoiseArray), label="H = 0.01 + Noise")
    plt.xlabel('X - Time')
    plt.ylabel('Y - X(t)')
    plt.title("Noise Values vs Actual")
    plt.legend(loc='best')
    plt.show()
    return


#sets the name of file,and starts step 2
def hvalues(Step,exactSolution):
    k,a, time, h, e, x, samples = step1(Step)
    u = []
    u.append(0)
    filename = str(h) + ".txt"
    filenames.append(filename)
    startfile(filename)
    if exactSolution == 0:
        x, time, u = step2(k, a, time, h, e, x, u, samples,filename)
    else:
        ExactSolution(k, a, time, h, e, x, u, samples,filename)
    return x,time,u
```

```
def Box_Muller(filename,ArrayX,TimeX,ArrayU0):
    #Step 0 Open file created earlier
    filename = filename + "+Noise.txt"
    data = ""
    XNoiseArray = []
    startfile(filename)
    # Step 1 set it=0, given µ, σ
    it, µ, σ = 0,0.0,0.001
    # Step 2 read record from file (time, x, u)
    counter=0
    if(len(ArrayX))== (len(TimeX))== (len(ArrayU0)):          #if all the same length
        while counter < len(ArrayX):
            Time = TimeX[counter]
            x = ArrayX[counter]
            u = ArrayU0[counter]
            #print("Time:",Time ," X:", x," U:" , u)
            # Step 3 if (it=0)
            if it==0:
                # Step 3a z1=rand (0, 2*pi)
                z1 = random.uniform(0, 2 * np.pi)
                # Step 3b b= σ * sqrt (-2*ln(rand(0,1))  "ln is natural log"
                #b = σ * np.sqrt(-2 * ln(random.uniform(0,1)))
                b = σ * np.sqrt(-2 * math.log(random.uniform(0,1)))
                # Step 3c z2= bsin(z1)+µ
                z2 = b * math.sin(z1) + µ
                # Step 3d  z3= bcos(z1)+µ
                z3 = b * math.cos(z1) + µ
                # Step 3e Xnoise=x+z2   Write to file (time, x, xnoise, u)
                Xnoise = x + z2
                # Step 3f it=1
                it=1
                counter+=1
            else:
                # Step 2g   it=0
                it=0
                # Step 2h   Xnoise=x+z3
                Xnoise = x + z3
                # Step 2i   Write to file (time, x, xnoise, u)
                counter +=1

            data +="\n"+"Time =" + str(Time) + " X ="+ str(x) + " Xnoise =" + str(Xnoise) + " U="+ str(u)
            XNoiseArray.append(Xnoise)
    writefile(data, filename)
            # Step 4  if not eof go to Step 2

    return XNoiseArray
```

```python
##################### PART 3
def plotExpected(noiseArray,expectedValues,part2Time,iterations,accstring):
    plt.plot(np.array(part2Time), np.array(expectedValues), label=" Perceptron Expected values")
    plt.plot(np.array(part2Time), np.array(noiseArray), label="H = 0.01 + Noise")
    plt.xlabel('X - Time')
    plt.ylabel('Y - X(t)')
    plt.title("Perceptron Values vs Actual, Iteration:" + str(iterations) +accstring )
    plt.legend(loc='best')
    plt.show()
    return


def Sigmoid(X,threshhold,U,T,i):
    #result = (U[i] - math.exp(-2*X))
    result = 3 / (1 + math.exp(-X))
    #result = U - math.exp((-2 * time[k]))
    return result


def initWeights():
    '''Initialize weights randomly'''
    inweight = []
    inthreshhold = uniform(-0.5, 0.5)
    for i in range(3):
        inweight.append(uniform(-0.5, 0.5))
    return inweight, inthreshhold


def activation(net_sum,inthreshhold):
    #print("Net Sum:", net_sum, " Threshhold:",inthreshhold)
    if net_sum < inthreshhold:
        GSerror = 1
    else:
        GSerror = 0
    print("Perceptron Output:",GSerror)
    return GSerror



def accuracy(target, error):
    expected = target + error
    if expected-target <0.1 and expected-target > -0.1 :
        return 1
    else:
        return 0
```

```python
#Iterates over each input and then calls activation to calculate an output
def SimPerceptron(x1,x2,x3,w1,w2,w3,inthreshhold,net_sum,U,Time,i):
    net_sum = 0
    '''Weighted sum of inputs = W1(X1) + W2(X2)+ W3'''
    BiasTerm= 0.5
    net_sum = x1*w1 +x2*w2 + x3*w3 - BiasTerm
    print("x1:",x1," x2:",x2, " x3:", x3,"Netsum:",net_sum)
    #Output = activation(net_sum,inthreshhold)
    Output = Sigmoid(net_sum,inthreshhold,U,Time,i)
    return Output , net_sum



##### Part 1
filenames = []
ArrayX0 ,ArrayTime0, ArrayU0 = hvalues(0.01,0)
ArrayX1 ,ArrayTime1, ArrayU1 = hvalues(0.25,0)
ArrayX2 ,ArrayTime2, ArrayU2 = hvalues(0.5,0)
ArrayX3 ,ArrayTime3, ArrayU3 = hvalues(0.75,0)
ArrayX4 ,ArrayTime4, ArrayU4 = hvalues(1,0)
ArrayX5 ,ArrayTime5, ArrayU4 = hvalues(0.01,1)
PlotAxis()

#### Part 2
Part2Array,Part2U, Part2Time = [],[],[]
for i in range(0,len(ArrayX0),10):
    Part2Array.append(ArrayX0[i])
    Part2U.append(ArrayU0[i])
    Part2Time.append(ArrayTime0[i])

NoiseArray = np.array(Box_Muller(filenames[0],Part2Array,Part2Time,Part2U))
plotNoise(NoiseArray,Part2Array ,Part2Time)

############# PART 3
weight, threshhold = initWeights()
Error = 1
Net_sum = 0
Iterations = 0
AccuArray = []
NoiseArray[:0] = 0
NoiseArray[:0] = 0
learning_rate = 0.2
```

```python
############# PART 3
weight, threshhold = initWeights()
Error = 1
Net_sum = 0
Iterations = 0
AccuArray = []
NoiseArray[:0] = 0
NoiseArray[:0] = 0
learning_rate = 0.2

#while Error*Error != 0 and Iterations < 101:
while Iterations < 2000:
    ExpectedValues = []
    ExpectedValues.append(0)
    ExpectedValues.append(0)
    for i in range(len(NoiseArray)-2):
        PerOutput, Net_sum = SimPerceptron(NoiseArray[i],NoiseArray[i+1],NoiseArray[i+2],
                                           weight[i],weight[i+1],weight[i+2], threshhold,
                                           Net_sum,Part2U,Part2Time,i)
        Target = NoiseArray[i+1]

        WDelta = Target - PerOutput
        Error = WDelta
        Wn = learning_rate * WDelta * NoiseArray[i]

        if len(weight) < len(NoiseArray):
            weight.append(Wn)
        else:
            weight[i] += Wn

        Expected = Target + Error
        ExpectedValues.append(Expected)
        Accuracy = accuracy(Target, Error)
        AccuArray.append(Accuracy)
        print("Accuracy :", int(Accuracy)," Error:", round(Error,6), "Target",
              round(Target,4),"Expected" ,round(Expected,4), "Net Sum:",round(Net_sum,4))

    print(" Iterations:", Iterations, "Accuracy :%", round(Accuracy,1))
    Iterations += 1
    Accstring = " Accuracy %"+ str( round(sum(AccuArray)/len(AccuArray)* 100,1 ))
    if Iterations== 1 or Iterations== 10 or Iterations== 50 or Iterations== 100 \
            or Iterations== 150 or Iterations== 200 or Iterations== 300 or Iterations== 400\
            or Iterations== 1000 or Iterations== 2000 :
        print(sum(AccuArray),"out of ",len(AccuArray))
        plotExpected(NoiseArray,ExpectedValues,Part2Time,Iterations,Accstring)
    AccuArray = []
```