CODING ASSIGNMENT 1

SOFT COMPUTING|| ME – 674

NAME: - Shaik Ahmad Ashraf Ali

ROLL.NO: - 214103006

DEPARTMENT: - MTech, Aerodynamics & Propulsion

CONTENT

1. Introduction 3
2. Problem Definition 3
3. Methodology 3
4. Data 5
5. Results 5
6. Conclusion 5
7. Code in C Language 9
    a. Variable Declaration 9
    b. Reading data from input …. 9
    c. Normalization of input and output data 11
    d. Initialisation of Connecting weights 12
    e. Computation of outputs of hidden and output neurons 12
    f. Mean Square Error calculations 14
    g. Weights updation by back propagation 14
    h. Testing cases 16

8. Reference 19

Figure-1 3
Figure-2 4
Table-1 5
Figure-3 6
Figure-4 6
Figure-5 7
Table-2 7

Introduction

In this report Artificial neural network was trained to validate the results of existing data. So that we can ensure the application of Artificial Neural Network for obtaining the results of various problem statements. Here we used the approach of Multilayer Feed Forward Neural Network (MLFFNN) so that our neural network can handle multiple inputs and multiple outputs with back propagation. Application of Artificial Neural Network was done to determine the penetration depth at high-velocity impact. Ballistic performance of armor was developed in minimum time with the help of hybrid method using simulation of FEM and Artificial Neural Network (ANN) to approximate ballistic limit thickness for armor steels.

Problem Definition:

Ballistic resistance against 7.62 mm armor piercing ammunition for high hard armor material is developed by a predictive model based on artificial neural network (ANN) is required. Here Multi- Layer Feed Forward Neural Network type of predictive model is used. Since it requires data to train the neural network, FEM simulations were used. In order to validate the results of FEM, few test cases were performed on 20 mm thick target with ballistic shots. After the training of ANN is done, now it is used to predict the ballistic limit thickness on 500 HB High Hard Steel Armor.

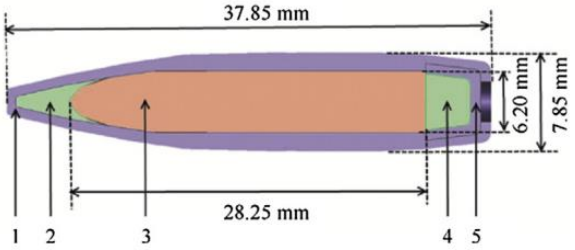

Fig-1: Cross-section View of 7.62 mm armor piercing ammunition
1-Brass jacket; 2- Lead antimony point filler; 3-Hardened steel core
4- Lead antimony base filler, 5- cup

Methodology:

A fully connected Multi-Layer feed forward Neural Network was chosen to predict the outputs of required problem statement. It has capability to access multiple inputs and obtain multiple outputs. With enough training, ANN will produce results of minimum error.

Here in this problem:

- Neural network consists of three layers, Namely Input layer with two input neurons and a hidden layer with required number of neurons for training and finally one output layer with single output neuron.
- Though the problem has two inputs and one output, the program was written in a generalized manner to accommodate any number of inputs and outputs.

- Target thickness (thickness of material used as a target) and incident velocity (m/s) (i.e. ammunition velocity) were the two inputs and depth (mm) of penetration in target material is the output.

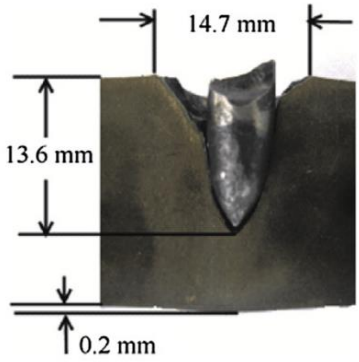

Fig-2: Section view of test showing penetration depth.

- We have chosen 15 data sets to train the Network and 4 data sets for testing the accuracy of obtained results from actual results.
- Here log-sigmoid (y = 1/(1+ e^(-x))) transfer function was used for both hidden layer and output layer.
- Data sets were normalized according to transfer function because, the range of log-sigmoid function was between 0 to 1.
- Normalization was required to make the connecting weights effective. Consider input values close to infinite, in this case connecting weight values will become close to 0, ineffective or invalid. Which makes program incompatible to solve the problem. Hence, normalization is required.
- Initial connecting weights for input – hidden (V_{ij}) and hidden – output (W_{jk}) were chosen at random using function rand() and sine function is used to limit the connecting weights between -1 to 1.
- In the research paper were tests were done in trial and error manner for optimum values of number of hidden neurons, learning rate and momentum term.
- End conditions for while loop was taken to be Mean Square Error (MSE) < 0.001 or number of iteration is less than 50,000.

Data:

Pattern	Target thickness (mm) (Input-1)	Incident velocity (m/s) (Input-2)	Penetration depth (mm) (Output)
1	9	450	6.68
2	9	600	9.00
3	10	450	6.58
4	10	600	9.81
5	12	450	6.32
6	12	600	9.16
7	15	450	6.33
8	15	600	8.72
9	15	750	11.5
10	18	450	6.38
11	18	600	8.45
12	18	750	10.83
13	12	750	11.95
14	20	450	6.39
15	20	600	8.36
16	20	750	10.80
17	20	854	12.90
18	18	854	12.97
19	15	854	13.00

Table-1: Datasets

The first fifteen rows of data were patterns used for training of "Artificial Neural Network" The last four rows were considered as Test Patterns for "Artificial Neural Network"

Results:

Here three graphs were plot for **"Mean Square Error Vs Iterations"** by considering learning rate values as 0.1, 0.3 and 0.5 and for each plot there are three curves each having certain number of hidden neurons and momentum term values.

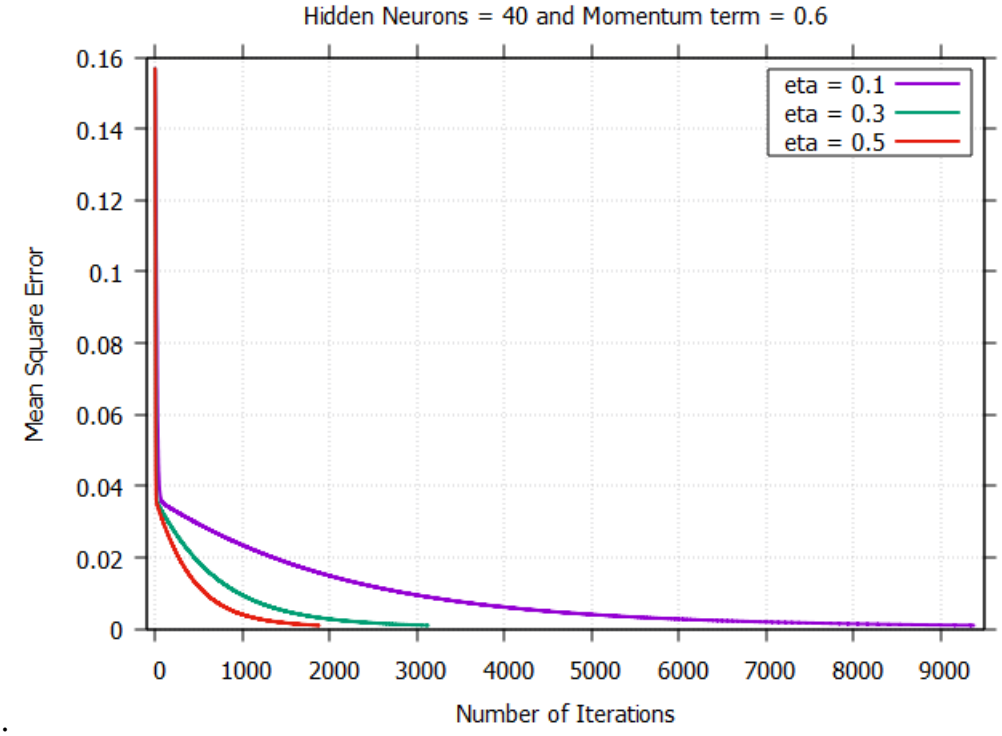

Fig-3: variation in mean square error with respect to learning rate for 40 hidden neurons and 0.6 as momentum coefficient term.

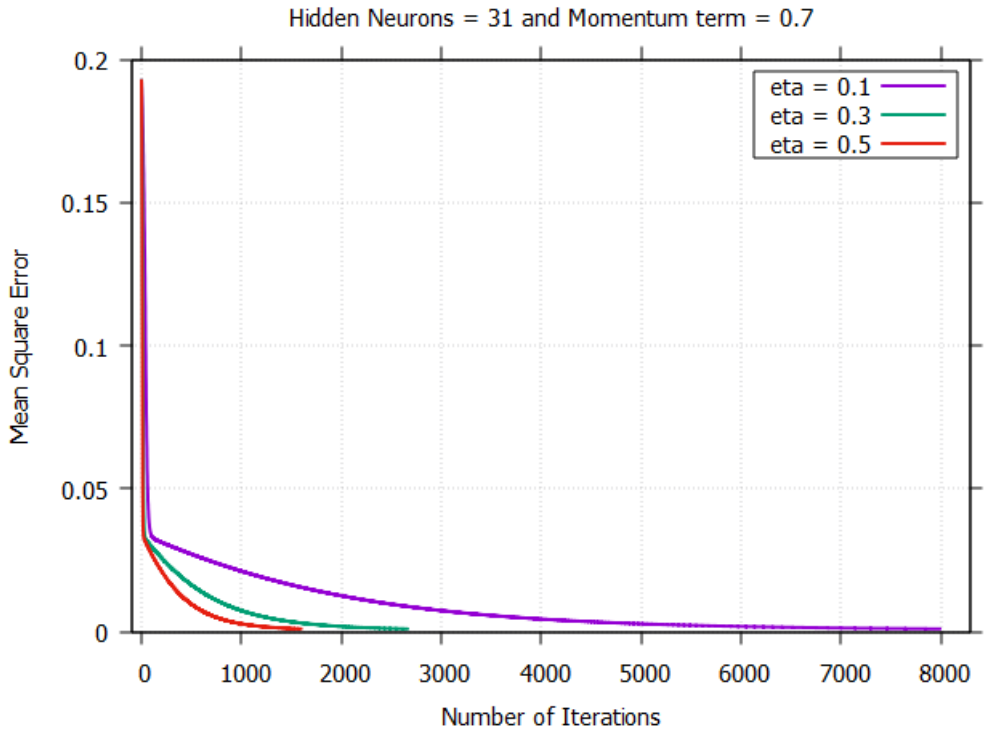

Fig-4: variation in mean square error with respect to learning rate for 31 hidden neurons and 0.7 as momentum coefficient term.

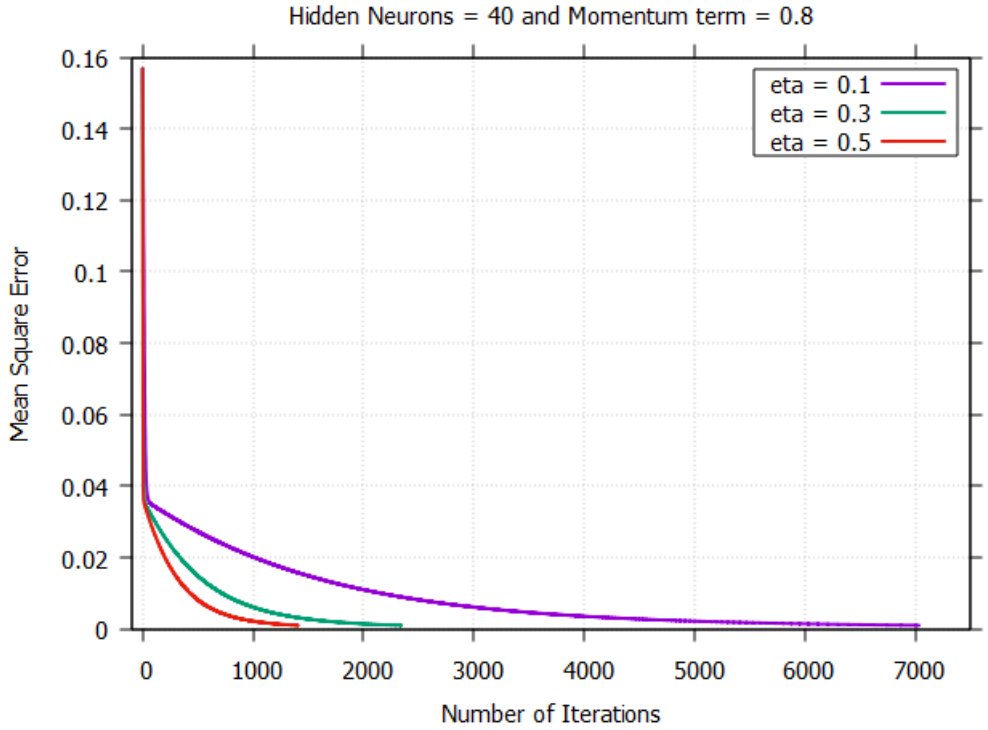

Fig-5: variation in mean square error with respect to learning rate for 40 hidden neurons and 0.6 as momentum coefficient term.

Pattern	**Target Output**	**Obtained Output Through ANN**								
		Learning rate = 0.1			**Learning rate = 0.3**			**Learning rate = 0.5**		
		A	**B**	**C**	**A**	**B**	**C**	**A**	**B**	**C**
16	10.8	9.964586	10.096272	9.964659	9.964725	10.096413	9.964844	9.964664	10.096729	9.965225
17	12.9	11.020627	11.217585	11.020737	11.020834	11.217773	11.021011	11.021039	11.218207	11.021582
18	12.97	11.167856	11.319994	11.167969	11.168066	11.320187	11.168248	11.168274	11.320635	11.168834
19	13	11.347494	11.444316	11.347611	11.347707	11.444511	11.347893	11.347917	11.444972	11.348494
Mean Square Error		0.004458	0.002879	0.004457	0.004456	0.002877	0.004453	0.004453	0.002873	0.004446
Number of Iteartions to reach MSE = 0.001		9376	7993	7033	3126	2665	2345	1876	1600	1408

A – Number of Hidden Neurons = 40; Momentum Term = 0.6

B – Number of Hidden Neurons = 31; Momentum Term = 0.7

C – Number of Hidden Neurons = 40; Momentum Term = 0.8

Table-2: output obtained for different properties with corresponding Mean Square Errors.

Conclusions:

From the above table, we can understand that,

- Increasing learning rate, decreases the number of iterations required to obtain Mean square error to be 0.001.
- From **A** and **C** in the table, for same number of hidden neurons, it shows that increasing momentum will decrease the number of iteration required to obtain 0.001 as a mean square error, and also decreases MSE obtained for test cases.
- From **A** and **B** in the table, it shows that, though there is a slight decrease in number of hidden neurons, solution may converge faster due to momentum term.
- From learning rates it is understandable that "Increasing learning rate may reduce number of iterations".
- Good change in MSE can be obtained by modifying Number of hidden neurons and momentum term.
- Considering Mean Square Error obtained in each case i.e. by varying Number of hidden neurons, learning rate and momentum term, MSE of final output was lower for 31 hidden neurons, 0.5 as learning rate and 0.7 as momentum term.
- Therefore, optimum values for the present problem statement was,

  Number of hidden neurons = 31
  Learning rate = 0.5
  Momentum Term = 0.7.

CODE in C Language:

Code for Multi-Layer Feed Forward Neural Network was done in C Language.

Code:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<stdlib.h>
int main()
{
        int i, j, k, p, L, N, P, T, TIO, M, count=0;
        /*TIO = total num of input and output neurons; T = test patterns;M = num of hidden neurons*/
        double a[100][100], V[100][100], W[100][100], E[100][100], dw[100][100], dv[100][100],
        eta, max, min, mse; /*E = Target - output*/
        double IH[100][100], OH[100][100], IO[100][100], OO[100][100], I[100][100],
        Ta[100][100], alpha; /* Ta = target matrix*/
        double Tmax[100], Tmin[100], AO[100][100];
          mse = 1;
          eta = 0.5;
          alpha = 0.7;
          M=31;
          FILE *ip,*op1,*op2;
          /*reading data from input*/
          ip=fopen("input_1.txt","r");
          op1=fopen("output_1.txt","w");
          op2=fopen("output_2.txt","w");
          fscanf(ip,"%d",&L);
          fscanf(ip,"%d",&N);
          fscanf(ip,"%d",&P);
          fscanf(ip,"%d",&T);
          fprintf(op1,"count                                    MSE\n");
          TIO = L+N;
          for(i=1;i<=P+T;i++)
          {
                for(j=1;j<=TIO;j++)
                {
                        fscanf(ip,"%lf",&a[i][j]);
```

```
        }
}
/*target matrix*/
for(k=L+1;k<=L+N;k++)
{
        for(p=1;p<=P+T;p++)
        {
                Ta[k-L][p] = a[p][k];
        }
}
fprintf(op2,"Target matrix : \n");
for(p=P+1;p<=P+T;p++)
{
        for(k=L+1;k<=L+N;k++)
        {
                fprintf(op2,"%lf ",Ta[k-L][p]);
        }
        fprintf(op2,"\n");
}
/*obtaining maximum and minimum matrices for targets*/
for(k=L+1;k<=L+N;k++)
{
        Tmax[k-L] = Ta[k-L][1];
        for(p=1;p<=P;p++)
        {
                if(Ta[k-L][p]>Tmax[k-L])
                {
                        Tmax[k-L] = Ta[k-L][p];
                }
        }
}
for(k=L+1;k<=L+N;k++)
{
        Tmin[k-L] = Ta[k-L][1];
        for(p=1;p<=P;p++)
        {
```

```
                if(Ta[k-L][p]<Tmin[k-L])
                {
                        Tmin[k-L] = Ta[k-L][p];
                }
        }
}
/*obtaining max and min from columns in datasets for normalization*/
for(j=1;j<=TIO;j++)
{
        max = a[1][j];
        min = a[1][j];
        for(i=1;i<=P+T;i++)
        {
                if(a[i][j]>max)
                {
                        max = a[i][j];
                }
                if(a[i][j]<min)
                {
                        min = a[i][j];
                }
        }
        /*normalising of datasets for log sigmoid function*/
        for(i=1;i<=P+T;i++)
        {
                a[i][j] = 0.1 + (0.8 * ((a[i][j]-min)/(max-min)));
        }
}
/*Input matrix*/
for(i=1;i<=L;i++)
{
        for(p=1;p<=P;p++)
        {
                I[i][p]=a[p][i];
        }
}
```

```
/*target matrix*/
for(k=L+1;k<=L+N;k++)
{
	for(p=1;p<=P+T;p++)
	{
		Ta[k-L][p] = a[p][k];
	}
}
/*random initialization of connecting weights*/
/*for V connecting weight b/w input and hidden*/
for(i=0;i<=L;i++)
{
	for(j=1;j<=M;j++)
	{
		V[i][j] = sin(rand());
	}
}
/*for W connecting weight b/w hidden and output*/
for(j=0;j<=M;j++)
{
	for(k=1;k<=N;k++)
	{
		W[j][k]= sin(rand());
	}
}
while(mse > 0.001 && count<50000)
{
	mse = 0;
	/*hidden input matrix*/
	for(p=1;p<P+1;p++)
	{
		for(j=1;j<=M;j++)
		{
			IH[j][p] = 0;
			for(i=1;i<=L;i++)
			{
```

```
                        IH[j][p]=IH[j][p] + (I[i][p] * V[i][j]);
                }
                IH[j][p] = IH[j][p] + (1*V[0][j]);
        }
}
/*hidden Transform matrix*/
for(p=1;p<P+1;p++)
{
        for(j=1;j<=M;j++)
        {
                OH[j][p] = 1/(1+exp(-IH[j][p]));
        }
}
/*o/p layer input matrix*/
for(p=1;p<P+1;p++)
{
        for(k=1;k<=N;k++)
        {
                IO[k][p] = 0;
                for(j=1;j<=M;j++)
                {
                        IO[k][p]=IO[k][p] + (OH[j][p] * W[j][k]);
                }
                IO[k][p] = IO[k][p] + (1*W[0][k]);
        }
}
/*output transformation matrix*/
for(p=1;p<P+1;p++)
{
        for(k=1;k<=N;k++)
        {
                OO[k][p] = 1/(1+exp(-IO[k][p]));
        }
}
```

```
/*Calculation of error*/
for(k=1;k<=N;k++)
{
        for(p=1;p<=P;p++)
        {
                E[k][p]=Ta[k][p]-OO[k][p];
                mse = mse + (0.5*E[k][p]*E[k][p]);
        }
}
mse = mse/P;
/*updating of W*/
for(j=0;j<=M;j++)
{
        for(k=1;k<=N;k++)
        {
                dw[j][k] = 0;
                for(p=1;p<=P;p++)
                {
                        dw[j][k] = dw[j][k] + ((E[k][p])*(OO[k][p])*(1-
                        OO[k][p])*(OH[j][p]));
                }
        }
}
for(j=0;j<=M;j++)
{
        for(k=1;k<=N;k++)
        {
                dw[j][k] = (eta/P)*dw[j][k];
        }
}
/*updating of V*/
for(i=0;i<=L;i++)
{
        for(j=1;j<=M;j++)
        {
                dv[i][j] = 0;
```

```
                for(p=1;p<=P;p++)
                {
                        for(k=1;k<=N;k++)
                        {
                                dv[i][j] = dv[i][j] + ((E[k][p])*(OO[k][p])*(1-
                                (OO[k][p]))*(W[j][k])*(OH[j][p])*(1-
                                OH[j][p])*(I[i][p]));
                        }
                }
        }
}
for(i=0;i<=L;i++)
{
        for(j=1;j<=M;j++)
        {
                dv[i][j] = (eta/(P*N))*dv[i][j];
        }
}
/*Wnew*/;
for(j=0;j<=M;j++)
{
        for(k=1;k<=N;k++)
        {
                W[j][k] = W[j][k] + (alpha * dw[j][k]);
        }
}
/*Vnew*/
for(i=0;i<=L;i++)
{
        for(j=1;j<=M;j++)
        {
                V[i][j] = V[i][j] + (alpha * dv[i][j]);
        }
}
count++;
printf(" count = %d and MSE = %lf \n\n",count,mse);
fprintf(op1,"%d                                %lf\n",count,mse);
```

```
}
printf("TEST CASE: \n");
/*printing of Vfinal*/
printf("V matrix is : \n");
for(i=0;i<=L;i++)
{
	for(j=1;j<=M;j++)
	{
		printf("%lf  ",V[i][j]);
	}
	printf("\n");
}
/*printing of Wfinal*/
printf("W matrix is : \n");
for(j=0;j<=M;j++)
{
	for(k=1;k<=N;k++)
	{
		printf("%lf  ",W[j][k]);
	}
	printf("\n");
}
/*test case inputs*/
for(i=1;i<=L;i++)
{
	for(p=P+1;p<=P+T;p++)
	{
		I[i][p]=a[p][i];
	}
}
/*hidden input matrix*/
	for(p=P+1;p<=P+T;p++)
	{
		for(j=1;j<=M;j++)
		{
			IH[j][p] = 0;
```

```
                for(i=1;i<=L;i++)
                {
                        IH[j][p]=IH[j][p] + (I[i][p] * V[i][j]);
                }
                IH[j][p] = IH[j][p] + (1*V[0][j]);
        }
}
/*hidden Transform matrix*/
for(p=P+1;p<=P+T;p++)
{
        for(j=1;j<=M;j++)
        {
                OH[j][p] = 1/(1+exp(-IH[j][p]));
        }
}
/*o/p layer input matrix*/
for(p=P+1;p<=P+T;p++)
{
        for(k=1;k<=N;k++)
        {
                IO[k][p] = 0;
                for(j=1;j<=M;j++)
                {
                        IO[k][p]=IO[k][p] + (OH[j][p] * W[j][k]);
                }
                IO[k][p] = IO[k][p] + (1*W[0][k]);
        }
}
/*output transformation matrix*/
for(p=P+1;p<=P+T;p++)
{
        for(k=1;k<=N;k++)
        {
                OO[k][p] = 1/(1+exp(-IO[k][p]));
        }
}
```

```
/*Actual output*/
for(k=L+1;k<=L+N;k++)
{
        for(p=P+1;p<=P+T;p++)
        {
                AO[k][p]= Tmin[k-L]+ ((Tmax[k-L]-Tmin[k-L])/0.8)*(OO[k-
                L][p]-0.1);
        }
}
printf("Actual output is: \n");
for(p=P+1;p<=P+T;p++)
{
        for(k=L+1;k<=L+N;k++)
        {
                printf("%lf ",AO[k][p]);
        }
        printf("\n");
}
fprintf(op2,"Actual output is: \n");
for(p=P+1;p<=P+T;p++)
{
        for(k=L+1;k<=L+N;k++)
        {
                fprintf(op2,"%lf ",AO[k][p]);
        }
        fprintf(op2,"\n");
}
fprintf(op2,"\n");
/*Calculation of error*/
for(k=1;k<=N;k++)
{
        for(p=P+1;p<=P+T;p++)
        {
                E[k][p]=Ta[k][p]-OO[k][p];
                mse = mse + (0.5*E[k][p]*E[k][p]);
        }
```

```
                }
                mse = mse/T;
                printf("mse = %lf \n",mse);
                fprintf(op2,"mse = %lf \n",mse);
                printf("error matrix: \n");
                for(k=1;k<=N;k++)
                {
                        for(p=P+1;p<=P+T;p++)
                        {
                                printf("%lf  ",E[k][p]);
                        }
                        printf("\n");
                }
        return 0;
}
```

Reference:

1. Determination of penetration depth at high velocity impact using finite element method and artificial neural network tools. Authors (Namık KILIÇ a, , Bu¨lent EKICI b , Selim HARTOMACIOGLU)