THE MONITORING OF A LATHE USING AN ARTIFICIAL NEURAL NETWORK – 5TH PART (RECORDINGS SPECTRAL ANALYSIS, USE OF ANN ON MONITORING OF THE TOOL WEAR)

George C. BALAN, Alexandru EPUREANU, Ciprian CUZMIN

Universitatea *Dunarea de Jos* din Galati
Str. Domneasca nr. 47 Galati – 800 008 ROMANIA
George.BALAN@ugal.ro

The study of machine-tool dynamic is realized here as "monitoring", meaning checking and improving the functioning of the machine. The state of processing is followed by certain sensors whose signs are processed inside the computer and then it takes the decision of monitoring, meaning the identification of a class from the set of classes (process conditions).
In this part of the paper there are presented recordings spectral analysis and use of ANN on monitoring of the tool wear.

Keywords: monitoring, lathe, spectral analysis, ANN

1. INTRODUCTION

In the first part of this paper there are presented the classes (tool conditions) for monitoring in turning and the artificial neural networks – **ANN**s (the creation of an ANN with the function *newff*).

In the second part of this paper there are presented the batch training of an artificial neural network with Levenberg-Marquardt algorithm and the experimental setup for components of cutting forces.

In the third part there are presented the experimental setup (for: cutter holder accelerations, cutting temperature, surface roughness, power), cutting working conditions and the tool wear.

In the fourth part there are presented the experimental results (validation of recordings) and the data processing (initial processing).

In this part of the paper there are presented recordings spectral analysis and use of ANN on monitoring of the tool wear.

2. RECORDINGS SPECTRAL ANALYSIS (DFT DISCRETE FOURIER TRANSFORM)

For length N input vector x, the DFT is a length N vector X, with elements

$$X(k) = \sum_{n=1}^{N} x(n)*\exp(-j*2*pi*(k-1)*(n-1) / N), \quad 1 <= k <= N.$$

The relationship between the DFT and the Fourier coefficients "a" and "b" in

$$x(n) = a0 + \sum_{k=1}^{N/2} (a(k)*\cos(2*pi*k*t(n) / (N*dt) + b(k)*\sin(2*pi*k*t(n) / (N*dt)))$$

is: $a0 = X(1) / N = (\sum_{n=1}^{N} x(n)) / N$, $a(k) = 2*real(X(k+1)) / N$, $b(k) = -2*imag(X(k+1)) / N$,

where x is a length N discrete signal sampled at times t with spacing dt. Remark: t(n) / dt = n.

To test the programme for the Fourier series, we apply it to the function:

x = 3 + 4*sin(2*pi*50*t) + 2*sin(2*pi*100*t) + 5*sin(2*pi*350*t);
Figure 1 shows the result to be correct.

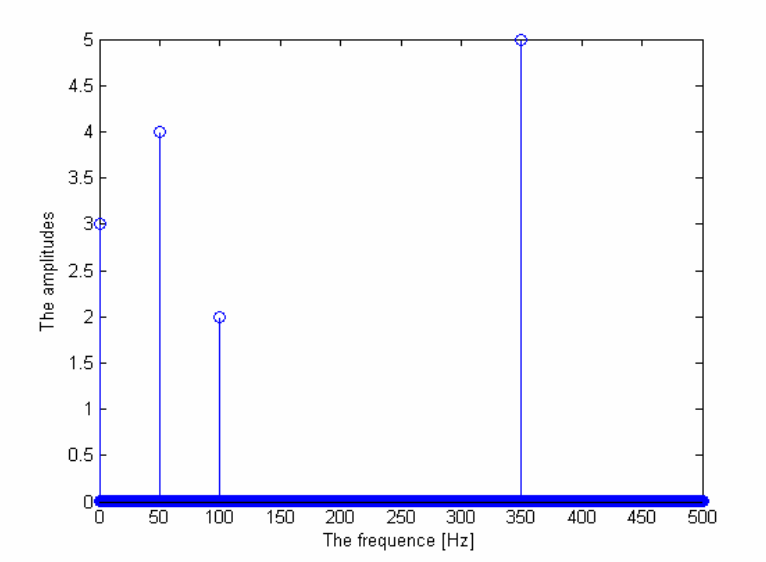

Fig. 1

This programme is further applied to recording no. 64 (belonging to class C6), for the spectral analysis of a_z and F_z. Figure 2 shows the graph.

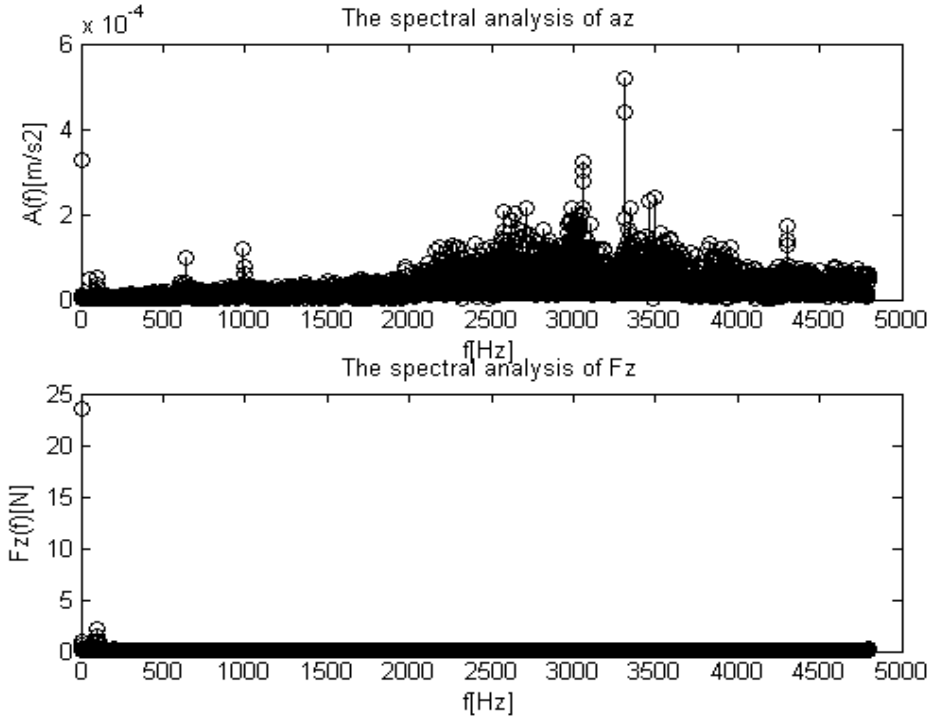

Fig. 2

With recordings in class C6 (see fig. 1 in [5]) F_z shows a periodical variation, for a long period: $T \approx 100.D\ t = 100.1/9600$, therefore with frequency $f = 1 / T = 96$ Hz, also present in figure 2, this being **the fundamental frequency** of the vibrations of the technological system.

The graphical representation $F_z = F_z\ (t)$ of recording no. 64 shows that vibration amplitude is of nearly *2.5* N, value which is sustainable in fig. 2.

3. USE OF ANN ON MONITORING OF THE TOOL WEAR

3.1. IMPROVING GENERALIZATION

One of the problems that occur during neural network training is called "over fitting". The error on the training set is driven to a very small value, but when new data is presented to the network the error is large. The network has memorized the training examples, but it has not learned to generalize to new situations.

One method for improving the network generalization is to use a network that is just large enough to provide an adequate fit. The larger a network you use, the more complex the functions the network can create. If we use a small enough network, it will not have enough power to over-fit the data.

Unfortunately, it is difficult to know beforehand how large a network should be
for a specific application. Other method for improving generalization is *early stopping*.

Note that if the number of parameters in the network is much smaller than the total number of points in the training set, then there is little or no chance of *over-fitting*. If you can easily collect more data and increase the size of the training set, then there is no need to worry about the following techniques to prevent over fitting. The rest of this section only applies to those situations in which you want to make the most of a limited supply of data.

EARLY STOPPING

Another method for improving generalization is called *early stopping*. In this technique, the available data is divided into three subsets. The first subset is the training set, which is used for computing the gradient and updating the network weights and biases. The second subset is the validation set. The error on the validation set is monitored during the training process. The validation error will normally decrease during the initial phase of training, as does the training set error. However, when the network begins to over fit the data, the error on the validation set will typically begin to rise. When the validation error increases for a specified number of iterations, the training is stopped, and the weights and biases at the minimum of the validation error are returned.

The test set (the third subset) error is not used during the training, but it is used to compare different models. It is also useful to plot the test set error during the training process. If the error in the test set reaches a minimum at a significantly different iteration number than the validation set error, this may indicate a poor division of the data set.

It is a good idea to train the network starting from several different initial conditions. It is possible for either method to fail in certain circumstances. By testing several different initial conditions, you can verify robust network performance.

3.2. PREPROCESSING AND POST PROCESSING

Neural network training can be made more efficient if certain pre-processing steps are performed on the network inputs and targets.

AVERAGE AND STAND. DEV. (prestd, poststd, trastd)

Before training, it is often useful to scale the inputs and targets so that they always fall within a specified range.

An approach for scaling network inputs and targets is to normalize the average and standard deviation of the training set. This procedure is implemented in the function *prestd*. It normalizes the inputs and targets so that they will have zero average and unity standard deviation. The following code illustrates the use of *prestd*: [pn,averagep,stdp,tn,averaget,stdt] = prestd(p,t);

The original network inputs and targets are given in the matrices "p" and "t". The normalized inputs and targets, "pn" and "tn", that are returned will have zero averages and unity standard deviation. The vectors "averagep" and "stdp" contain the average and standard deviations of the original inputs, and the vectors "averaget" and "stdt" contain the averages and standard deviations of the original targets. After the network has been trained, these vectors should be used to transform any future inputs that are applied to the network. They effectively become a part of the network, just like the network weights and biases.

If *prestd* is used to scale both the inputs and targets, then the output of the network is trained to produce outputs with zero average and unity standard deviation. If you want to convert these outputs back into the same units that were used for the original targets, then you should use the routine "poststd". In the

following code, we simulate the network that was trained in the previous code, and then convert the network output back into the original units.

an = sim(net,pn);
a = poststd(an,averaget,stdt);

The network output "an" corresponds to the normalized targets "tn". The un-normalized network output "a" is in the same units as the original targets "t".

If *prestd* is used to pre-process the training set data, then whenever the trained network is used with new inputs, they should be pre-processed with the averages and standard deviations that were computed for the training set. This can be accomplished with the routine *trastd*.

PRINCIPAL COMPONENT ANALYSIS (prepca, trapca)

In some situations, the dimension of the input vector is large, but the components of the vectors are highly correlated (redundant). It is useful in this situation to reduce the dimension of the input vectors. An effective procedure for performing this operation is *principal component analysis*. This technique has three effects: it orthogonalizes the components of the input vectors (so that they are uncorrelated with each other); it orders the resulting orthogonal components (principal components) so that those with the largest variation come first; and it eliminates those components that contribute the least to the variation in the data set. The following code illustrates the use of "prepca", which performs a principal component analysis.

[pn,averagep,stdp] = prestd(p);
[ptrans,transMat] = prepca(pn,0.02);

Note that we first normalize the input vectors, using "prestd", so that they have zero average and unity variance. This is a standard procedure when using principal components. In this example, the second argument passed to "prepca" is 0.02. This averages that "prepca" eliminates those principal components that contribute less than 2% to the total variation in the data set. The matrix "ptrans" contains the transformed input vectors. The matrix "transMat" contains the principal component transformation matrix. After the network has been trained, this matrix should be used to transform any future inputs that are applied to the network. It effectively becomes a part of the network, just like the network weights and biases. If you multiply the normalized input vectors "pn" by the transformation matrix "transMat", you obtain the transformed input vectors "ptrans".

If "prepca" is used to pre-process the training set data, then whenever the trained network is used with new inputs, they should be pre-processed with the transformation matrix that was computed for the training set. This can be accomplished with the routine "trapca".

3.2- A N N

The techniques described in chapters 3.1 and 3.2 will be applied in the following.

The training set will contain the recordings in the "Learning" set, therefore 60% of the total number of recordings. Recordings "b" in the "Classification" set will be allotted to the validation set, and recordings "d" will be input into the testing set; therefore each set has 20 % of the recordings.

ANN consists of *3* levels, $s_1 = 23$ neurons, $s_2 = 27$, $s_3 = 7$ (= number of classes). The input matrix *p* has the dimensions *12* (monitoring indices) *x 655* (recordings), and the output matrix *y* has dimensions *7* (classes) *x 655*. The training functions are: *tf1 = purelin, tf2 = tansig, tf3 = logsig*, therefore the output vectors have *7* elements, with values in domain (0, 1).

In the sequence with instructions:

[ma, ia]=max(a); [mt, it]=max(t); c = ia = = it; n = 1: 655; nc = [n, c];

mt = 1 (obviously), and "ma" is the maximum value of the "a" column; if indices "ia" and "it" (which give the position of the respective maximum values) are identical, then the maximum values on the "a" columns coincide with the maximum values on the "t" columns and within matrix *c (1x655)* the respective elements equals *1*, and in an opposite case it will take the value *0*. "nc" is a matrix with two rows, the first one counts the recordings, while the second has the elements *1* (correct), or *0* (error).

The first runs (with the training functions *trainrp, trainscg,* etc.) showed errors only in the positions corresponding to the recordings in classes *c5* and *c7*, i. e. in the case of those classes which have the fewest recordings. The increase of the number of recordings – without making new experiments – may be performed by adding the same recordings several times to the same class, which may be eventually affected by a noise of an average value of *0.1*; for example, if the recordings are included in the table "Table 112", featuring *960 x 6*, then the recordings:

Table 112 + randn(960,6)*0.1 will be added.

Consequently:

- we fourfold the recordings in class *c5*: an average value noise of *0.1* is attached to the first set, the second set is identical with the original one, an average value noise of *0.15* will be attached to the third set;
- we threefold the recordings in class *c7*, acting by a way of analogy with the foregoing (only with first and the second set).

To sum up, the classes in chapter 4b have their final form as follows:

- recordings 43 ÷ 63, 77 ÷ 87, i. e. 32 Rec. (with 4800 samples each) fit into class "c_1" ($VB_{max} \le 0{,}2$ mm);
- recordings 89 ÷ 91, 94, 96 ÷ 112 $\Rightarrow$ 21 Rec. $\rightarrow$ class "c_2" ($0{,}2 < VB_{max} \le 0{,}4$ mm);
- 113 ÷ 118, 124 ÷ 130, 132 ÷ 144, 146 ÷ 163 $\Rightarrow$ 44 Rec. $\rightarrow$ "c_3" ($0{,}4 < VB_{max} \le 0{,}6$ mm);
- 164 ÷ 171, 174 ÷ 177 $\Rightarrow$ 12 Rec. $\rightarrow$ "c_4" ($0{,}6 < VB_{max} \le 0{,}7$ mm);
- 182 ÷ 184 $\Rightarrow$ 3 Rec. + 3 Rec. (with noise) + 3 Rec. + 3 Rec. (with noise) = 12 Rec. $\rightarrow$ "c_5" ($VB_{max} > 0{,}7$ mm);
- 64, 67, 72 ÷ 76, 93, 95, 131, 172, 179, 180, 185 $\Rightarrow$ 14 Rec. $\rightarrow$ "c_6" (Chatters);
- 187 ÷ 191 $\Rightarrow$ 5 Rec. + 5 Rec. (with noise) + 5 Rec. $\Rightarrow$ 15 Rec. $\rightarrow$ "c_7" (Idle turning).

Therefore, *c5* will consist of *60* recordings, *c7 – 75* recordings, and the number of columns in the above matrices grows from *655* to *750*.

With instructions: ind = find(c = = 0); dim = length(ind); er = dim / 750,
we find: the "c" elements indices which are null, the "ind" number of elements as well as the network error, respectively.

The results of a run are presented in what follows: R = 8 ; Q = 750

There was a redundancy in the data set, since the principal component analysis has reduced the size of the input vectors from *12* to *8*.

```
TRAINRP, Epoch 0/300, MSE 1.53359/0, Gradient 0.292509/1e-006
TRAINRP, Epoch 25/300, MSE 0.487294/0, Gradient 0.0150939/1e-006
TRAINRP, Epoch 43/300, MSE 0.477332/0, Gradient 0.0148044/1e-006
TRAINRP, Validation stop.
ind =
  Columns 1 through 18
   440   441   445   446   447   448   449   450   454   455   456   457   458   459   463   468   472   473
  Columns 19 through 37
   474   475   476   477   481   482   483   519   520   521   533   545   594   595   596   646   647   658   672
```

dim = 37 ; er = dim / 750 = 0.0493 = 4.93 %.

It is a useful diagnostic tool to plot the training, validation and test errors to check the progress of training. The result is shown in the figure 3.

The result here is reasonable, since the test set error and the validation set error have similar characteristics.

In other runs:

- carried out under the same circumstances, results were twice as above, and once as follows: 51epochs; dim = 27 ; er = 3.6 %;
- without "init" function (to reinitialize weights and biases), in two runs the errors were 4.67% and 13.3%;
- with the *trainscg* training function without "init" the error was 5.33%, whereas with "init"- 25.6%.

4. ACKNOWLEDGEMENT

This research was supported by a grant CNCSIS [1] and a contract CEEX [6].

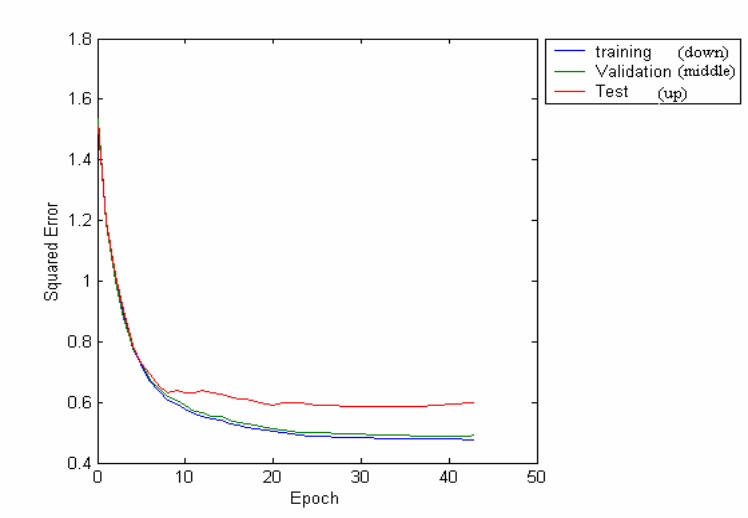

Fig. 3

REFERENCES

1. BALAN, G., *Monitorizarea unui strung utilizand o retea neuronala artificiala*, Contract (grant tip A) nr 33 445 / '02, Tema 19, Cod CNCSIS 451.
2. BALAN, G., EPUREANU, A., 2004, *The monitoring of a lathe using an artificial neural network – 1-st part (Theoretical basis)*, The Annual Symposium of the Institute of Solid Mechanics SISOM 2004, Academia Romana, Bucuresti, pe CD-ul Simpozionului.
3. BALAN, C. G., EPUREANU, A., POPESCU, F., 2005, *The monitoring of a lathe using an artificial neural network – 2nd part (the training of an artificial neural network and the experimental setup)*, The Annual Symposium of the Institute of Solid Mechanics SISOM 2005, Academia Romana, Bucuresti, pe CD-ul Simpozionului.
4. BALAN, C. G., EPUREANU, A., VACARUS, V., 2006, *The monitoring of a lathe using an artificial neural network – 3 rd part (the experimental setup)*, The Annual Symposium of the Institute of Solid Mechanics SISOM 2006, Academia Romana, Bucuresti, pe CD-ul Simpozionului.
5. BALAN, C. G., 2006, *The monitoring of a lathe using an artificial neural network – 4th part (experimental results, data processing)*, The Annual Symposium of the Institute of Solid Mechanics SISOM 2006, Academia Romana, Bucuresti, pe CD-ul Simpozionului.
6. EPUREANU, A., 2005, *Simulation, modeling and virtual production methods based on informatics and communication technologies*, Contract no. 22 CEEX I03/'05, MEdC,