

COMPARING DIFFERENT IMPLEMENTATIONS FOR THE LEVENBERG-MARQUARDT ALGORITHM

Dário Baptista, Fernando Morgado-Dias

*Madeira Interactive Technologies Institute and Centro de Competências de Ciências Exactas e da Engenharia,
Universidade da Madeira
Campus da Penteada, 9000-039 Funchal, Madeira, Portugal.
Tel: +351 291-705150/1, Fax: +351 291-705199*

Abstract: Artificial Neural Networks are used in a variety of problems occurring either in research or in the industry. The first step is to train a network to perform a desired function, which requires a training algorithm. Levenberg-Marquardt is a second order algorithm which outperforms Backpropagation and is currently available in most Neural Network toolboxes. This paper tests two toolboxes, Neural Network Toolbox of MatLab and Neural Network System Identification Toolbox, in order to demonstrate that the implementations differ according to the toolbox used and that Matlab obtains better results in all the datasets used. This paper also explains the differences between the implementations of each tool and the advantages/disadvantages of each toolbox. Copyright CONTROLO2012

Keywords: Artificial Neural Networks, Training Algorithms, Early Stopping, Levenberg-Marquardt, Backpropagation.

1. INTRODUCTION

Artificial Neural Networks are used in a variety of problems occurring either in research or in the industry. The first step for building an Artificial Neural Network (ANN) consists of training the network to perform a certain task. This requires a training algorithm. Backpropagation is probably the most diffused algorithm in ANN but the Levenberg-Marquardt (LM) is recognized as achieving a much higher performance namely, by converging more often and by making training faster.

The LM algorithm has, nevertheless, a few details regarding its implementation that deserve further attention. It is also uncertain why the most common tool used for training ANN, Matlab, has not had a Levenberg-Marquardt version in its toolbox for several years.

The first LM version diffused was in a toolbox freely distributed (Nørgaard, 1996a), which was a result of a PhD thesis (Nørgaard, 1996b) concluded in 1996. Only in the following decade did Matlab propose a LM version in its Neural Network toolbox.

This work investigates whether the two implementations are alike and tests them in a few sets of data using supervised learning to determine which one performs better.

2. ANN STRUCTURE

In this work, a Multilayer Feed-Forward Artificial Neural Network (FANN) was chosen with an AutoRegressive with eXogenous signal (ARX) model to represent the different systems. A FANN is composed of an input layer, one or more hidden layers and an output layer. Figure 1 shows a Multilayer Feed-Forward ANN with only one hidden layer.

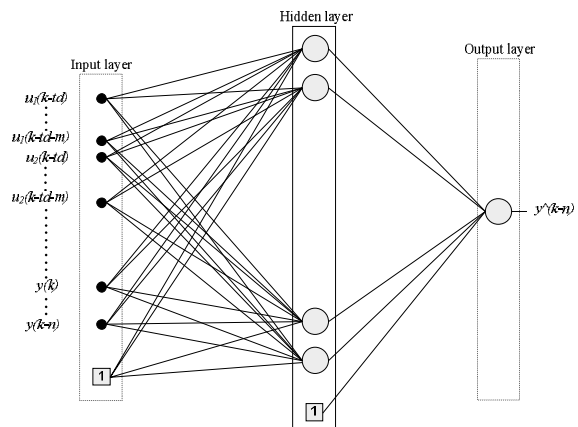


Fig. 1. Multilayer Feed-Forward ANN structure.

The basic element of a FANN is the neuron, which is a logical-mathematical structure that seeks to simulate the shape, behavior and functions of a biological neuron (Morgado-Dias *et al.*, 2006). Figure 2 shows schematically the neuron structure.

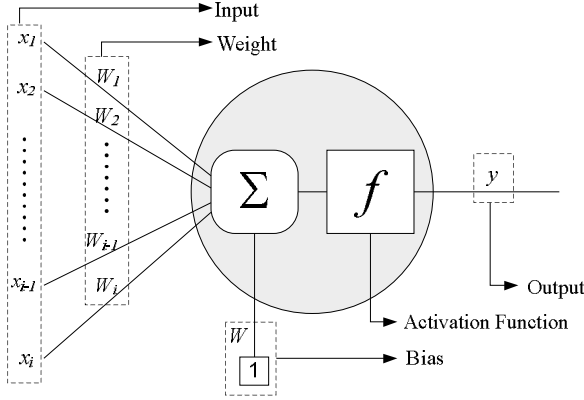


Fig. 2. Neuron Structure.

The input information is captured and is processed by the sum function, and the new signal generated by the neuron is set by the activation function (Wright *et al.*, 2007). Hence, the neurons are based on a simple mathematical function which can be translated analytically by the following form (Morgado *et al.*, 2006):

$$y = f\left(\sum_{i=1}^N I_i w_i\right) \quad (1)$$

where, I_i is the i^{th} input of the network, w_i is the corresponding weight and f is the activation function. Thus, the Multi Input Single Output FANN in figure 1 implements the following equation (Morgado *et al.*, 2006):

$$y = f_1\left(\sum_{i=1}^{nh} w_{i1} f_i\left(\sum_{k=1}^{ni} I_{ki} w_k\right)\right) \quad (2)$$

where nh is the number of neurons in the hidden layer, ni is the number of inputs, f_i is the activation function of the hidden layer and f_1 is the activation function of the output layer. In this work the hyperbolic tangent is used in the neurons of the hidden layer and the linear function in the neuron of output layer (Morgado *et al.*, 2006).

3. LEVENBERG-MARQUARDT ALGORITHM

The training process is based on minimizing an error function, in each iteration, such as the one in equation (3):

$$F(x_k) = \frac{1}{N} \sum_{i=1}^N v_i(x_k)^2 \quad (3)$$

where N is the number of samples used to train the FANN; x_k is the vector of parameters, in this case, the set of weights at iteration k ; $v_i(x_k) = o_i - y_i(x_k)$, o_i is

the i^{th} desired output for the sample, and $y_i(x_k)$ is the i^{th} FANN output during iteration k .

Starting from the Taylor series approach of second order, the following can be written (Morgado-Dias *et al.*, 2006):

$$F(x_{k+1}) = F(x_k + \Delta x_k) \approx F(x_k) + G(x_k)\Delta x_k + \frac{1}{2}\Delta x_k^T H(x_k) \Delta x_k \quad (4)$$

where, $\Delta x_k = x_{k+1} - x_k$, $H(x_k)$ is the Hessian matrix of $F(x_k)$ and $G(x_k)$ is the gradient of $F(x_k)$. If the derivative of equation 3 in respect to Δx_k is taken, we obtain:

$$G(x_k) + H(x_k)\Delta x_k = 0 \quad (5)$$

$$\Leftrightarrow \Delta x_k = -H(x_k)^{-1} G(x_k)$$

The Gradient and the Hessian matrix may be written in the following form:

$$G(x_k) = 2J^T(x_k)v(x_k) \quad (6)$$

$$H(x_k) = 2J^T(x_k)J(x_k) + 2S(x_k) \quad (7)$$

where $J(x_k)$ is the Jacobian and $S(x_k)$ is:

$$S(x_k) = \sum_{i=1}^N v_i(x_k) \frac{\partial^2 v_i(x_k)}{\partial x_{k_i} \partial x_{k_i}} \quad (8)$$

where x_{k_i} is the weight at position i , in iteration k and x_{k_i} is the weight at position i , in iteration k . These indexes were not used before to avoid complexity in the notation.

If it can be assumed that $S(x_k)$ is small when compared to the product of the Jacobian, then the Hessian matrix can be approximated by the following:

$$H(x_k) \approx 2J^T(x_k)J(x_k) \quad (9)$$

Thus, expression (4) could be written in the following form:

$$\Delta x_k = -[2J^T(x_k)J(x_k)]^{-1}(2J^T(x_k)v(x_k)) \quad (10)$$

One limitation which could occur with this algorithm is that the simplified Hessian matrix might not be invertible. To overcome this problem a modified Hessian matrix may be used:

$$Hm(x_k) = H(x_k) + \mu I \quad (11)$$

where I is the identity matrix and μ is a value such which makes $Hm(x_k)$ positive definite and which therefore is invertible. Thus, expression (9) may be re-written in the following form:

$$\Delta x_k = -\frac{(2J^T(x_k)v(x_k))}{2J^T(x_k)J(x_k) + \mu_k I} \quad (12)$$

algorithm. This algorithm is a result of the independent work of both authors: Levenberg and Marquardt (Levenberg, 1944) (Marquardt, 1963).

This last change in the Hessian matrix corresponds to the updating rule for the Levenberg-Marquardt

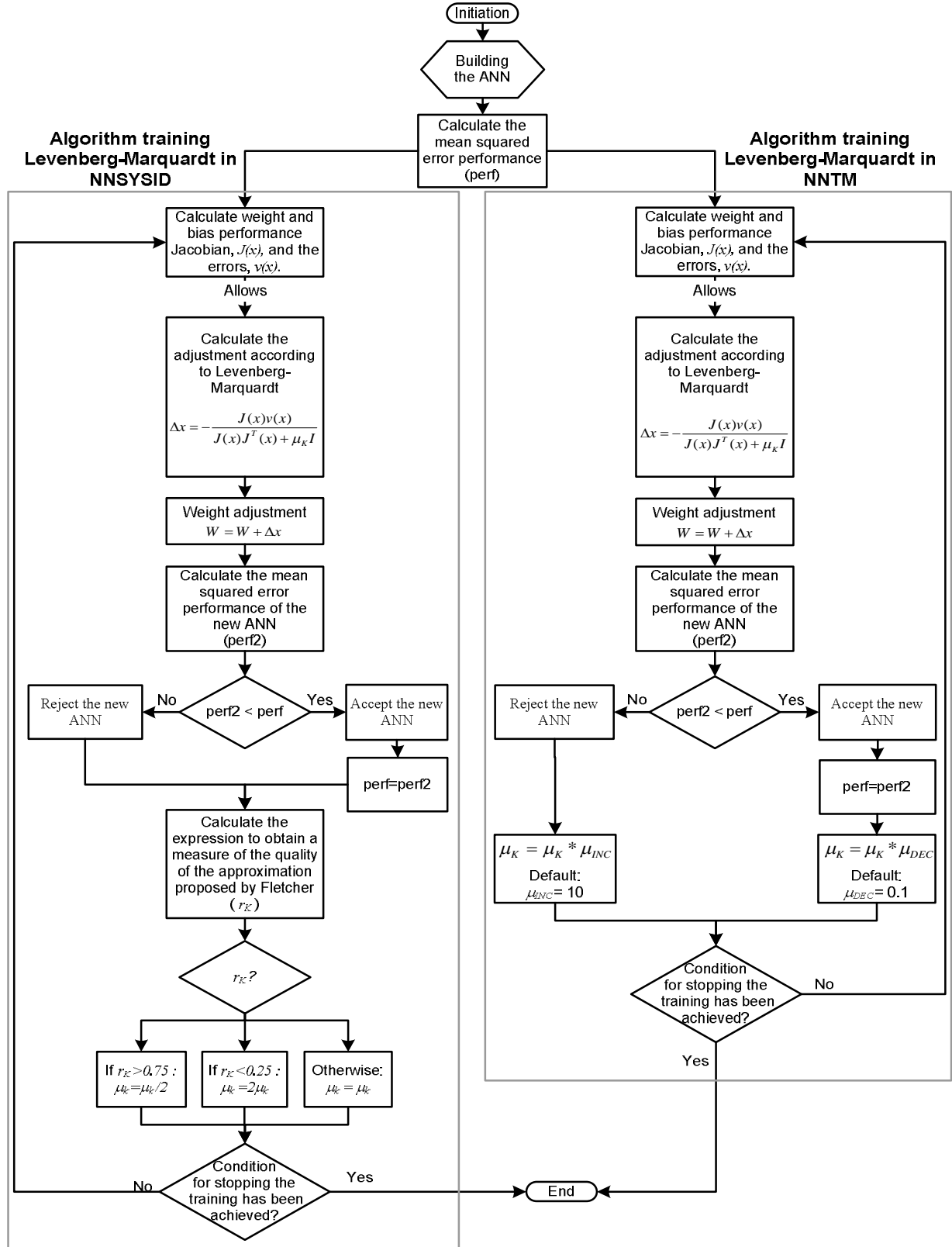


Fig. 3. Levenberg-Marquardt algorithm implementations in NNSYSID and NNTM.

The parameter μ_k is introduced in the Hessian matrix (10) to make it positive-definite so that it can be inverted. The selection of μ_k , the “step” of the LM is of vital importance for the functioning and efficiency of the algorithm, because it is responsible for stability as well as the speed of convergence. It is in the selection of the value of μ_k that differentiates the implementations used in the Neural Network Toolbox of Matlab (NNTM) and the Neural Network System Identification toolbox (NNSYSID).

Figure 3 is a block diagram of both implementations which illustrates the differences of the LM algorithm’s implementations in both tools. The left side of figure 3 was explained in (Dias *et al.*, 2006) and the right side of the same figure was created analysing the Matlab source code.

As can be seen from figure 3, after building the ANN with a single output and multiple inputs (MISO - Multi Input Single Output) and before beginning the training, it is necessary to obtain information about the performance index of the ANN. In both cases, the performance index will be based on the MSE.

Then, the next iteration is calculated according to LM, using eq.3, and each weight is adjusted, resulting in a new iteration.

For this new iteration the MSE has to be calculated and compared with the previous iteration. If the new performance is lower than the previous one, the new iteration will be accepted; otherwise, it will be rejected.

However, the way in which the parameter μ_k is chosen between both tools is different. On the one hand, an inflexible method is used in the NNTM. This means that if the new iteration was to be accepted, μ_k would be decremented; otherwise, μ_k would be incremented. On the other hand, in the NNSYSID tool the solution proposed by Fletcher (Nørgaard, 1996b), shown in eq. 4, is used, in order to obtain a measure of the quality of approximation (Morgado *et al.*, 2006).

$$r_k = \frac{V_N(x_k) - V_N(x_k + f_k)}{V_N(x_k) - L_N(x_k + f_k)} \quad (13)$$

where V_N is the function to be minimized and L_k is the estimate of that value calculated from the Taylor series of second order and f_k is the search direction, in the present situation, the search direction given by the Levenberg-Marquardt algorithm.

This approximation is only valid in a neighbourhood of the current iteration. This neighbourhood diminishes as μ_k increases. Equation 13 is used as a measure of the quality of the approach and is used to increase and decrease μ_k , according to figure 3.

4. TEST RESULTS

To compare both Levenberg-Marquardt implementations, models for three different datasets were built. The initial set of weights is the same for the ANN trained with NNTM and NNSYSID and is changed in each experiment. The datasets used are:

pH neutralization process in a stirring tank (De Moor, 2010a), ball-and-beam (De Moor, 2010b), and wing flutter (De Moor, 2010c), freely available SISTA’s Identification.

Models were built for ANN with 3, 6, 10, 15 and 20 neurons (10 of each kind) for each dataset (a total of 150 ANN). During training, the behavior of MSE is a monotonous decreasing function on the training sequence. However, the behavior of the test sequence is different. In the first iterations the MSE decreases, corresponding to a phase where the network is learning the main features of the system. At a certain point the MSE begins to grow, corresponding to a greater influence of variance error (Sjöberg *et al.*, 1992). At this stage the network is learning characteristics of the noise or the training sequence. To avoid this problem we have used the equivalent to Early Stopping (Sjöberg *et al.*, 1992) and kept the ANN at the point where it exhibits the best performance.

Figures 4, 5 and 6 show the average of the results obtained with both tools for each dataset as a function of the number of neurons.

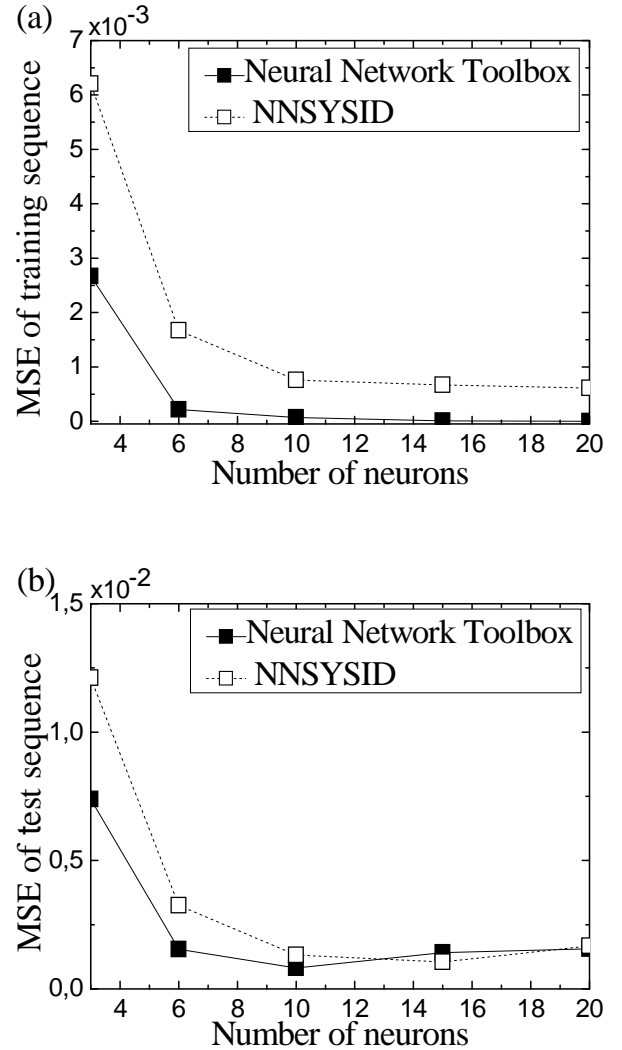


Fig 4. pH neutralization process in a stirring tank. (a) MSE of training sequence; (b) MSE of test sequence.

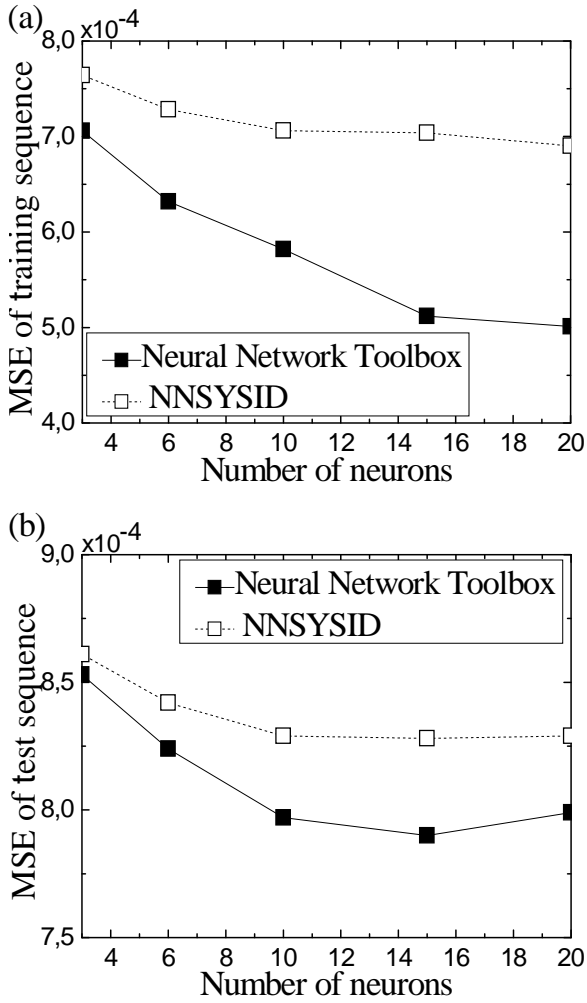


Fig 5. Ball-and-beam. (a) MSE of training sequence; (b) MSE of test sequence.

Both tools show MSE errors of train and test sequence in the same order of magnitude, although the training error of MATLAB is always smaller. Regarding the test error the values obtained with both toolboxes are very close, the largest difference for the best architecture being of 6,7%. For some sizes of the ANN the NNSYSID performs better than the NNTM, namely when the network is larger, but in most situations NNTM achieves a better result. In the Wing flutter test, the best result is so similar that it is not easy to distinguish from figure 6b). The best is from NNTM with $5,95E-05$ using 10 neurons while the NNSYSID obtained $5,98E-05$ using 20 neurons.

5. CONCLUSIONS

In this work the implementation of the Levenberg-Marquardt algorithm in two different tools, NNTM and NNSYSID was studied. The difference between the tools is the choice of parameter μ_k . NNTM uses an inflexible method, i.e., the value μ_k is increased or decreased according to the latest iteration. On the other hand, the NNSYSID calculates the value μ_k using a solution proposed by Fletcher to obtain a measure of the quality of approximation and decide the length of the step.

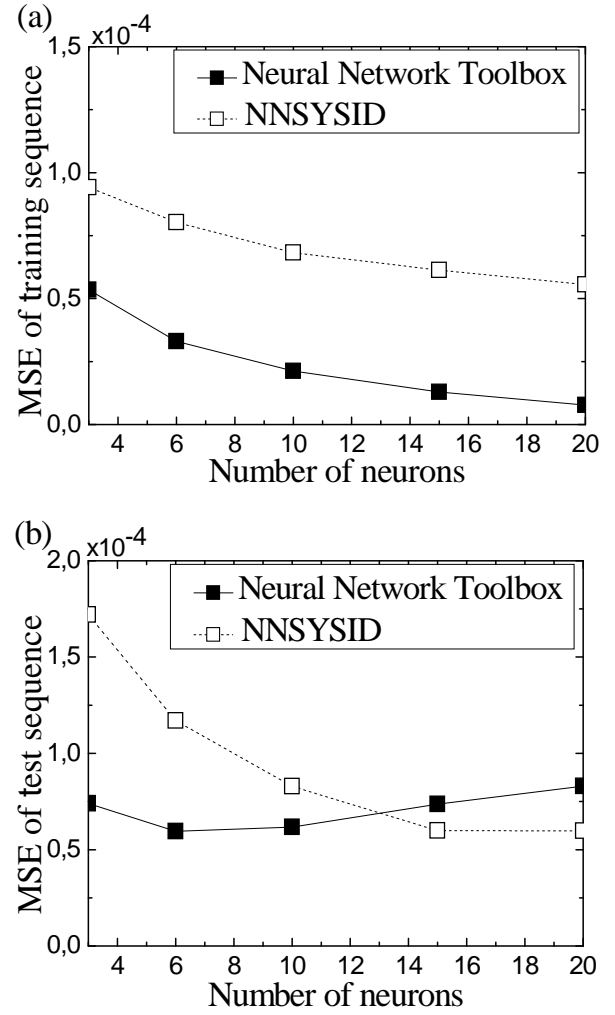


Fig 6. Wing flutter. (a) MSE of training sequence; (b) MSE of test sequence.

From the training and test results it can be concluded, with the sets used, that the NNTM implementation of the Levenberg-Marquardt performs better than NNSYSID. Nevertheless it should be pointed out that while the NNTM is paid as an extra to MATLAB, NNSYSID can be downloaded and used without cost. When using both tools it becomes evident that the structure used for simulating ANN in MATLAB is much more complex and difficult to use than the one proposed in NNSYSID.

ACKNOWLEDGMENTS

The authors would like to acknowledge the Portuguese Foundation for Science and Technology for their support for this work through project Pest-OE/EEI/LA0009/2011.

REFERENCES

- Nørgaard, M. (1996a). Neural Network System Identification Toolbox for MATLAB. Technical Report.
- Nørgaard, M. (1996b). System Identification and Control with Neural Networks. PhD Thesis,

Department of Automation, Technical University of Denmark.

- Dias F. M., Antunes A., Vieira J., Mota A. (2006). A sliding window solution for the on-line implementation of the Levenberg-Marquardt algorithm. In: Engineering Applications of Artificial Intelligence, Vol. 19/1, pp. 1-17, IFAC.
- Wright S., Marwala, T. (2007). Artificial Intelligence Techniques for Steam Generator Modelling. School of Electrical and Information Engineering, P/Bag x3, Wits, South Africa
- Levenberg K. (1944). A method for the solution of certain problems in least squares. Quart. Appl. Math., Vol. 2, pp. 164—168.
- Marquardt D. (1963). An algorithm for least -squares estimation of nonlinear parameters. SIAM J. Appl. Math., Vol. 11, pp 431—441.
- De Moor B.L.R. (ed.) (2010a), DaISy: Database for the Identification of Systems, Department of Electrical Engineering, ESAT/SISTA, K.U.Leuven, Belgium, URL:<http://homes.esat.kuleuven.be/~smc/daisy/>, 7 Dec 2010. [Used dataset: Simulation data of a pH neutralization process in a stirring tank, section Process Industry Systems, 96-014.]
- De Moor B.L.R. (ed.) (2010b), DaISy: Database for the Identification of Systems, Department of Electrical Engineering, ESAT/SISTA, K.U.Leuven, Belgium. URL:<http://homes.esat.kuleuven.be/~smc/daisy/>, 7 Dec 2010. [Used dataset: Data of the ball-and-beam setup in SISTA, section Mechanical Systems, code 96-004].
- De Moor B.L.R. (ed.) (2010c), DaISy: Database for the Identification of Systems, Department of Electrical Engineering, ESAT/SISTA, K.U.Leuven, Belgium, URL:<http://homes.esat.kuleuven.be/~smc/daisy/>, 7 Dec 2010. [Used dataset: Wing flutter data, section Mechanical Systems, code 96-008].
- Sjöberg J., Ljung L. (1992). Overtraining, Regularization and Searching for minimum in Neural Networks, Preprint IFAC Symp. on Adaptive Systems in Control and Signal Processing, pp. 669 – 674, Grenoble, France.