15th Triennial World Congress, Barcelona, Spain

NEURAL PREDICTIVE CONTROL TOOLBOX FOR CACSD IN MATLAB ENVIRONMENT

Jesús M. Zamarreño

Dpt. System Engineering and Automatic Control. University of Valladolid

Abstract: This paper describes a neural predictive control toolbox developed in Matlab/Simulink environment. The application permits all phases of the system design: simulation of the plant by means of any Simulink model, loading of input/output data, definition of the neural network architecture, training, and, finally, application of the predictive control strategy based on the neural network model. *Copyright © 2002 IFAC*

Keywords: Computer-aided control systems design, Predictive control, Model-based control, Neural control, Neural networks.

1. INTRODUCTION

Control of complex industrial processes like MIMO (Multiple Input Multiple Output) processes, processes with long and variable dead time, or processes with significant non-linearities is not well resolved with the classical PID. It is necessary to employ other techniques that can deal with this class of processes. One of the most extended control techniques, in particular in petrochemical processes, is the Model-based predictive control (MBPC). MBPC methods have been shown to be an effective tool for reducing operating costs, one of the main aims in industry, and have been playing a very important role in the area of process control (Richalet, 1993; Camacho and Bordons, 1995) due to their ability to handle difficult control problems which involve multivariable process interactions, constraints in the system variables, time delays, etc.

The classical MBPC algorithms use linear models of the process to predict the output of the process over a certain horizon, and to evaluate a future sequence of control signals in order to minimize a certain cost function that takes account of the future output prediction errors over a reference trajectory, as well as control efforts (Clarke, *et al.*, 1987; Garcia, *et al.*, 1989). When no model of the system is available, the classical system-identification theory (Ljung, 1987) provides possible solutions to the problem, but when the process is non-linear and it is driven over a wide dynamic operating range, the use of linear models becomes impractical, and the identification of non-linear models for control becomes a necessity.

In recent years, the use of neural networks for nonlinear system identification has proved to be extremely successful. In fact, the idea of incorporating neural-network models in MBPC algorithms has been proposed by a number of researchers (Bhat and McAvoy, 1989; Hunt, *et al.*, 1992; Williams, 1990; Zamarreño and Vega, 1999). In these cases, the neural model has been embedded into the MBPC scheme providing a neural predictive controller (NPC).

NPC has been applied not only in simulation, but also in real industrial processes (Zamarreño and Vega, 1997). For successful application to a real process, a tool for control system design would be helpful for tuning and configuration of the controller, besides development of the model. Among the computer-based environments for control system design, perhaps one of the most popular is Matlab, from The Mathworks, Inc.

This paper describes a computer-aided control systems design (CACSD) for NPC. The NPC algorithm employed was first described by the author in (Zamarreño and Vega, 1996) and the neural network for obtaining the predictions in the MBPC scheme is called state-space neural network (ssNN) (Zamarreño and Vega, 1998).

This paper begins with a description of the NPC and the way in which it is implemented. Next, the NPC toolbox is described in detail, explaining the steps that must be carried out. Finally, the paper ends with some conclusions.

2. NEURAL PREDICTIVE CONTROL

Neural predictive control is basically a type of model-based predictive control, where the model for predictions is a neural network. Fig. 1 represents the block scheme of the control configuration. The different parts involved in the neural predictive control are explained below.

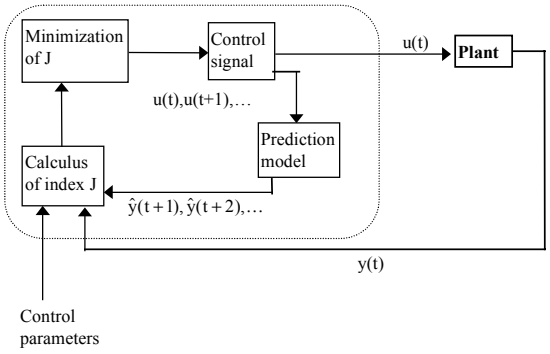


Fig. 1. Block scheme of the model-based predictive control configuration.

2.1 Algorithm description

The model predictive control (MPC) strategy (De Keyser, 1991) is based on the use of a model to predict the future output trajectory of the process. Next, the algorithm computes the future control actions to minimize a performance index at each sampling time *t*, and the first control action is applied. The procedure is repeated at time *t + 1*.

Optimization problem. The control evaluation consists of the minimization problem of the following general index, which is formulated in such a way that MIMO systems can be dealt with, as well as systems in which the number of manipulated variables is not equal to the number of controlled variables:

$$J = \sum_{k=1}^{dim-y} \eta_k \cdot \gamma_k \cdot \sum_{j=N1_k}^{N2_k} (\hat{y}_k(t+j) - ref_k(t+j))^2 + \\ + \sum_{k=1}^{dim-u} \sigma_k \beta_k \cdot \sum_{j=1}^{Nu_k} (u_k(t+j-1) - u_k(t+j-2))^2 \quad (1)$$

The minimization of index J is performed subject to the following restrictions:

a) Restrictions on the value of the manipulated variable:

$$Umin_k \le u_k(t+j-1) \le Umax_k \\ j = 1,..,Nu_k \; ; k = 1,...,dim\text{-}u \quad (2)$$

b) Restrictions on the variation of the manipulated variable:

$$\Delta Umin_k \le u_k(t+j-1) - u_k(t+j-2) \le \Delta Umax_k \\ j = 1,..,Nu_k \; ; k = 1,...,dim\text{-}u \quad (3)$$

c) Restrictions on the future values of the outputs:

$$Ymin_k \le \hat{y}_k(t+j) \le Ymax_k \\ j = N3_k,..,N4_k \; ; k = 1,...,dim\text{-}y \quad (4)$$

d) Restrictions on the final value of the manipulated variable:

$$\Delta u_k(t+j) = 0 \\ j \ge Nu_k \; ; k = 1,...,dim-u \quad (5)$$

where u_k are the control variables, $\hat{y}_k$, the model prediction and ref_k the reference trajectory which is defined, as is usual in MPC, by a first-order system with time constant α_k, that drives the system from the present point to the desired set-point sp_k, i.e.,

$$ref_k = \frac{\alpha_k}{1-(1-\alpha_k)z^{-1}} sp_k \quad (6)$$

Other parameters are as follows:

- γ_k and β_k: weighting factors for each output and input, respectively.
- η_k and σ_k: the output and input availability factors. Possible discrete values are {0,1}. If, for any reason, the field measurements do not reach the controller, they are zero.
- $N1_k$ and $N2_k$: the initial and the prediction horizons for each output.
- Nu_k: the control horizon for each input.
- $Umin_k$, $Umax_k$, $\Delta Umin_k$ and $\Delta Umax_k$: limits for the control actions and control increments, respectively.
- $Ymin_k$, $Ymax_k$: minimum and maximum values of the output between the horizons defined by $N3_k$ and $N4_k$.

Note that the status of every variable is taken into account through a set of weighting factors that modify the optimization problem accordingly, making the NPC very flexible. On-line reconfiguration is possible. This reconfiguration issue is of great interest in the event of auto/manual operations, when one output is not available, in case of changes from controlled to limited status, etc. All these characteristics are included in the toolbox.

Non-linear effects are introduced into the problem formulation through the expression chosen for $\hat{y}_k$ and by means of the constraints. The predictions are functions, represented by a state-space neural network, of the past and future values of the control variables and future values of the disturbances **d**. The control problem can be seen as a standard non-linear programming (NLP) problem that will be solved in the NPC toolbox by means of the *constr* function from the optimization toolbox.

The control signal at each sampling time t is the solution to the previous optimization problem. The solution gives the control signals $u_k(t)$, $u_k(t+1),\ldots,u_k(t+Nu_k-1)$ (k=1,…,dim-u), but only the first one, $u_k(t)$, is actually applied to the plant. As the function to be minimized is non-linear and very complex, no analytical solution is given; the problem is solved numerically at each sampling time.

2.2 Modelling and non-linear predictions evaluation

A deterministic non-linear system can be represented in discrete state-space form as:

$$\mathbf{x}(k+1) = f_1(\mathbf{x}(k), \mathbf{u}(k), \mathbf{d}(k)) \\ \mathbf{y}(k) = f_2(\mathbf{x}(k)) \quad (7)$$

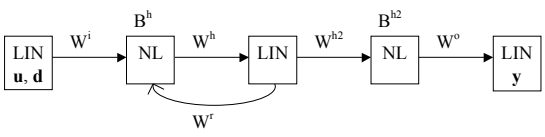
where $\mathbf{x} \in \Re^s$ is the state vector, $\mathbf{u} \in \Re^{n-d}$ is the control input vector, $\mathbf{d} \in \Re^d$ is the measurable disturbance vector, $\mathbf{y} \in \Re^m$ is the output vector, and f_1: $\Re^s \times \Re^n \to \Re^s$ and f_2: $\Re^s \to \Re^m$ are two static nonlinear mappings.

It is well known from the neural-network field that it is possible to represent any arbitrary non-linear function by means of a neural network with one hidden layer consisting of non-linear elements (Hornik et al., 1989). Thus, the proposed ssNN architecture, with suitable hidden layers to obtain f_1 and f_2, can be actually used to represent (7):

$$\begin{aligned} \hat{\mathbf{x}}(t) &= \mathbf{W^h} \cdot f1(\mathbf{W^r} \cdot \hat{\mathbf{x}}(t-1) + \mathbf{W^i} \cdot \begin{pmatrix} \mathbf{u}(t-1) \\ \mathbf{d}(t-1) \end{pmatrix} + \mathbf{B^h}) \\ \hat{\mathbf{y}}(t) &= \mathbf{W^o} \cdot f2(\mathbf{W^{h2}} \cdot \hat{\mathbf{x}}(t) + \mathbf{B^{h2}}) \end{aligned} \quad (8)$$

where the first equation takes into account the possible nonlinear dynamics of the states, and the second is a nonlinear mapping from the states to the output. This NN can be represented as a block diagram (Fig. 2). The parameters W and B are properly dimensioned matrices and vectors, respectively, which must be estimated to fit the ssNN and the real plant response, given an ordered temporal sequence for the inputs. The sum squared error (SSE) between the real output and the simulated one can be used as a criterion to adjust the weights and bias in the NN using a gradient-based technique (Hoekstra and Kooijman, 1991). The gradient can be explicitly evaluated and, consequently, the convergence speed improved. However, better results were obtained when training experiments were performed using a random search technique, such as the modified random search optimization method (Baba, 1989) and this technique was used in this toolbox for training the ssNN.

The use of this model to generate predictions is more or less straightforward. The way to proceed is to let the dynamical system evolve a number of steps equal to the desired number of predictions. All of the predictions from the system are obtained from the actual state of the system (given by $\hat{\mathbf{x}}$), the actual and future values of **u** (the last vector produced by the optimization algorithm) and the actual and future values of the disturbances (in the actual version, future values are assumed constant). Whenever the controller is called (at every sampling time), the ssNN must obtain the next state in order to keep up to date when the predictions are needed.



LIN: Linear Processing Elements
NL: Non-linear Processing Elements (Sigmoid)

Fig. 2. State-space neural network.

3. THE NPC TOOLBOX

For NPC design, a CACSD is convenient. This allows to simulate the process, training of the neural network based on input/output data collection, and tuning of the parameters of the controller to obtain an appropriate response. The tool that allows all these has been developed in the Matlab/Simulink environment.

As it has been said before, the application has been divided into five steps:

1. Definition of the model to be controlled and inputs for identification purposes.
2. Normalization of the input/output data for feeding in to the neural network.
3. Definition and training of the ssNN neural network.
4. Definition of a steady state for the model representing the plant, to become the initial condition for simulation.
5. Definition of the NPC parameters and application of the control algorithm in simulation.

In the following, a description of each of the steps is given.

3.1 Simulink model and inputs

The first step to be accomplished (Fig. 3) is to select the simulink model that will represent the plant to be controlled. The toolbox automatically detects the number of inputs and number of outputs as well as their names if the simulink model is well structured with inputs and outputs ports.

Other parameters to be entered in this window are the simulation time for data generation to be used at the identification phase as well as the sampling time to be used.

Finally, for the inputs provided by the simulink model, one has to select if they are manipulated inputs or disturbances. Whatever the type of input, for generation of the outputs through the model, the inputs must be selected. There are several types of inputs:

- From an ASCII file
- Random, gaussian signal
- Random, binary signal
- Pseudorandom, binary signal
- Sum-of-sinusoid signal
- Protocol
- Sinusoid
- Constant

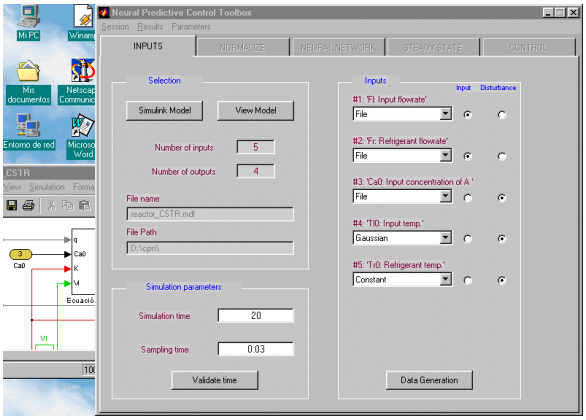


Fig. 3. First step: Model and data generation.

3.2 Normalization of the data

Neural network training is best carried out if the data has been normalized. At this stage, one has to enter appropriate limits for the inputs and outputs, so the toolbox can perform the mapping of these intervals to the interval (-1,1).

3.3 Neural network architecture and training

The toolbox is designed to provide various kinds of neural networks architectures and various training algorithms, but right now there is only one possibility for choosing: ssNN for the neural network architecture and a modified random optimization method for the training algorithm. In future versions, this will be augmented with other architectures and algorithms.

As the ssNN is composed of five layers, as explained in section 2.2, one has to enter the number of neurons (Fig. 4) for first and second hidden layer, as well as for the state layer. Besides, as neurons contained in hidden layers may have nonlinear transfer functions, one has to select what kind of function they have.

Another parameter is the data interval to be used for the training algorithm. This has two goals: first, as ssNN is a recurrent network, it is possible that the initial dynamic behaviour is different from the model, so the first samples should not be considered; and second, maybe one desires to preserve some of the data for validation purposes.

Finally, the parameters for the training algorithm must be provided. Once it is done, one has to push the "Train new network" button, so the training procedure begins. Evolution of the error along the training epochs is plotted, so one can see if training is being successful. Another possibility, once the algorithm has finished, is to compare the data with the response of the ssNN (Fig. 5). It is possible, that, after that, re-training is necessary. In this case, there is button labelled "Train again" that continues the training from the last set of weights obtained.

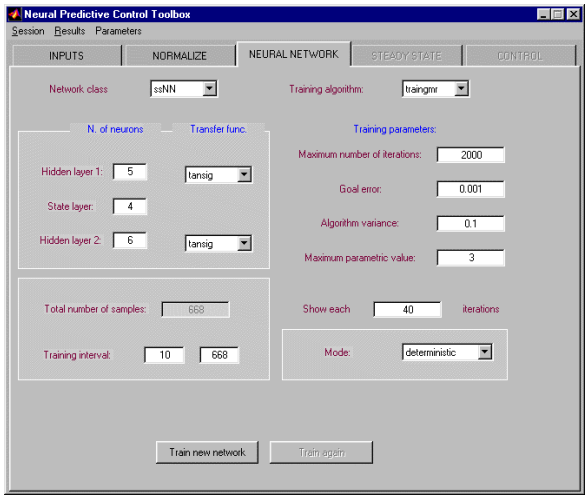


Fig. 4. Third step: definition of the neural network architecture and parameters of the training algorithm.

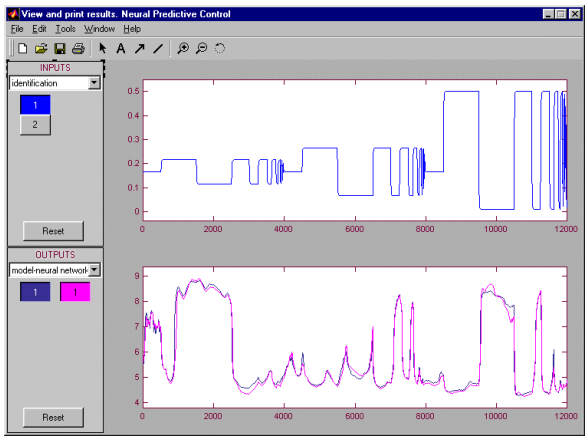


Fig. 5. Identification inputs (above) and comparison between the real outputs and ssNN outputs (below).

3.4 Steady state as initial condition

For applying the control algorithm to the simulink model, it is desirable that the plant be in steady state at the beginning of the simulation. This is the goal of the fourth step of the toolbox. As the steady state of the model depends on the value of the inputs, the user has to introduce the value of every input as well as the desired accuracy of the steady state and the maximum number of iterations. With these data, the toolbox simulates the model until the steady state is reached.

3.5 Control

The last step is the application of the neural predictive controller based on the predictions provided by the ssNN previously designed to the simulink model representing the plant.

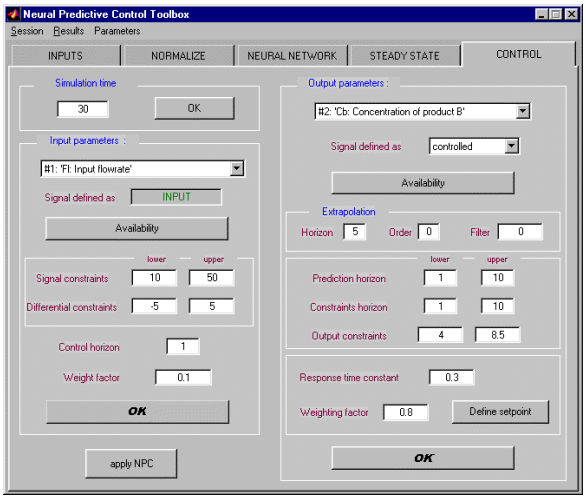


Fig. 6. Fifth step: NPC parameters.

All the parameters (Fig. 6) explained in section 2.1 have to be provided for every manipulated variable and every controlled or constrained output. The functionality is complete, including the possibility of simulating faults in sensors or actuators through the availability buttons. The set-point for every controlled output can be chosen with the number of steps desired by the user. Once all the parameters have been introduced, the user can run the simulation in closed loop.

3.6 Exploiting the results

The toolbox provides another possibilities like plotting and saving to disk several signals like input/output data for the identification, neural network response to the same pattern that the model, manipulated variables provided by the NPC, set-point, and, of course, evolution of controlled variables (Fig. 7).

There are two stages where the toolbox may require significant calculation time: training of the neural network and NPC application. In both cases, the optimization problem is time consuming. For flexibility, the user can save the current session and load it afterwards.

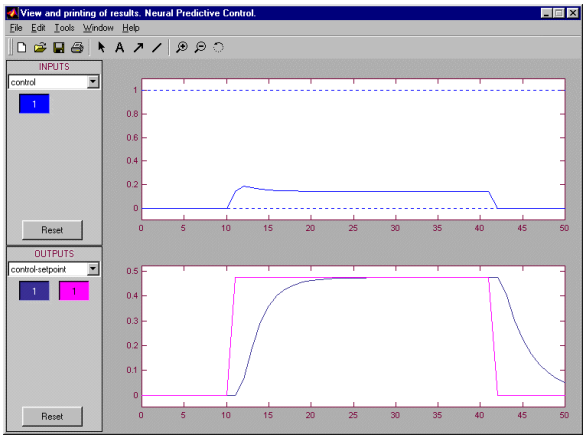


Fig. 7. Viewing results of the control algorithm.

4. COMPARISON WITH OTHER SIMILAR TOOLS

There exist other neural predictive controllers implemented as m-functions in Matlab. For example, version 4.0 of the Neural Network Toolbox (Demouth and Beale, 2001) implements a Simulink block called NN Predictive Controller. This block gives access to the identification phase, but only the identification of a neural network ARX model is possible. Another limitation of the controller is that N1 is fixed at 1 and only SISO models can be controlled with this block, which is a severe constraint since the main applicability of predictive control in industry is for multivariable processes.

Another toolkit called NNCTRL toolkit (Norgaard, 2000) also includes a Nonlinear Generalized Predictive Control based on Neural Networks. This toolkit is an add-on to the NNSYSID toolbox, which is a toolbox for system identification with neural networks. The toolkit does not provide a graphical user interface for designing the controller. Identification and control phases are accomplished independently, though the neural network model obtained at the identification phase can be used in the controller directly. For configuration of the controller, the user has to modify an initialization ASCII file which acts as a framework where the user writes the design parameters, experiment definition, etc. With respect to the predictive controller implemented in the toolkit, there are two versions called npccon1 which uses a Quasi-Newton method for optimization, and npccon2 which uses a Newton-based Levenberg-Marquardt method. The toolkit is intended to be used on SISO processes which can be a handicap in the application to a multivariable process.

In summary, the neural predictive control toolbox presented in this paper is flexible enough to accommodate other neural networks models and/or training algorithms, is easy to use through the graphical user interface and can be applied to MIMO processes. This last characteristic cannot be found in other similar products, as known by the author.

4. CONCLUSION

This paper has described a neural predictive control toolbox[1] developed in the Matlab/Simulink environment. The toolbox provides an easy way to do the design and validation of the NPC through simulation. The interface is divided in five windows that match the five steps required to design the NPC. The interface is easy to use and intuitive.

Programming of the application had in mind the possibility of improving its characteristics. In particular, adding a new neural network architecture

[1] Download at http://www.isa.cie.uva.es/productos/

or a new training algorithm will be easy. This will be accomplished as future work.

ACKNOWLEDGEMENTS

The author wants to express his gratitude to Julio Fernández and Andrés Mañanes for their work in the Matlab programming, and also to the support obtained from CICYT through the project PPQ2000-1075-C02-01.

REFERENCES

Baba, N. (1989). A new approach for finding the global minimum of error function of neural networks. *Neural Networks*, **2**, 367-373.

Bhat, N., T.J. McAvoy (1989). Use of neural network for dynamic modelling and control of chemical process systems. *American Control Conference*, Pittsburg.

Camacho, E.F. and C. Bordons (1995). *Model predictive control in the process industry*. Springer, London.

Clarke, D.W., C. Mohtadi, P.S. Tuffs (1987). Generalized predictive control – part I. The basic algorithm; Part II. Extensions and interpretations. *Automatica*, **23**, 137-160.

De Keyser, R. (1991). Basic Principles of MBPC. *European Control Conference*, Grenoble.

Demuth H., M. Beale (2001). Neural Network Toolbox User's Guide. The MatWorks, Inc.

Garcia, C.E., D.M. Prett, M. Morari (1989). Model predictive control: theory and practice – a survey. *Automatica*, **25**, 335-348.

Hoekstra, J., R. Kooijman (1991). Recurrence with delayed links in multilayer networks for processing sequential data. *International Conference on Artificial Neural Networks*, Espoo.

Hornik, K., M. Stinchcombe, H. White (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**, 359-366.

Hunt, K.J., D. Sbarbaro, R. Zbikowski, P.J. Gawthrop (1992). Neural networks for control systems – a survey. *Automatica*, **28**, 1083-1112.

Ljung, L. (1987). *System identification. Theory for the user*. Prentice Hall, New Jersey.

Norgaard M. (2000). Neural Network Based Control System Design Toolkit, ver. 2. Technical Report 00-E-892, Department of Automation, Technical University of Denmark.

Richalet, J. (1993). Industrial applications of model based predictive control. *Automatica*, **29**, 1251-1274.

Williams, R.J. (1990). Adaptive state representation and estimation using recurrent connectionist networks. In: *Neural Networks for Control* (W.T. I.I.I. Miller, R.S. Sutton, P.J. Werbos (Ed)), 97-114. The MIT Press, Cambridge, MA.

Zamarreño J.M., P. Vega (1996). Neural predictive control. Application to a highly non-linear system. *13th World Congress International Federation of Automatic Control*, San Francisco.

Zamarreño J.M., P. Vega (1997). Identification and predictive control of a melter unit used in the sugar industry. *Artificial Intelligence in Engineering*, **11**, 365-373.

Zamarreño J.M., P. Vega (1998). State space neural network. Properties and application. *Neural Networks*, **11**, 1099-1112.

Zamarreño J.M., P. Vega (1999). Neural predictive control. Application to a highly non-linear system. *Engineering Applications of Artificial Intelligence*, **12**, 149-158.