Generalized Predictive Control and Neural Generalized Predictive Control

Sadhana CHIDRAWAR[1], Balasaheb PATRE[1]

Asst. Prof, MGM's College of Engineering, Nanded (MS) 431 602
Professor, S.G.G.S. Institute of Engineering And Technology, Nanded (MS) 431 606

E-mail(s): sadhana_kc@rediff.com. bmpatre@yahoo.com

Abstract

As Model Predictive Control (MPC) relies on the predictive Control using a multilayer feed forward network as the plants linear model is presented. In using Newton-Raphson as the optimization algorithm, the number of iterations needed for convergence is significantly reduced from other techniques. This paper presents a detailed derivation of the Generalized Predictive Control and Neural Generalized Predictive Control with Newton-Raphson as minimization algorithm. Taking three separate systems, performances of the system has been tested. Simulation results show the effect of neural network on Generalized Predictive Control. The performance comparison of this three system configurations has been given in terms of ISE and IAE.

Keywords

Neural network; Model predictive control; GPC; NGPC.

Introduction

In recent years, the requirements for the quality of automatic control in the process industries increased significantly due to the increased complexity of the plants and sharper specifications of product quality. At the same time, the available computing power increased to a very high level. As a result, computer models that are computationally expensive became

applicable even to rather complex problems. Intelligent and model based control techniques were developed to obtain tighter control for such applications.

In recent years, incorporation of neural networks as intelligent control techniques, to adaptive control system design has been claimed to be a new method for the control of systems with significant nonlinearities. Those neural network based control systems so far developed are generally classified as indirect or direct control methods.

Model predictive control (MPC) has found a wide range of applications in the process, chemical, food processing and paper industries. Some of the most popular MPC algorithms that found a wide acceptance in industry are Dynamic Matrix Control (DMC), Model Algorithmic Control (MAC), Predictive Functional Control (PFC), Extended Prediction Self Adaptive Control (EPSAC), Extended Horizon Adaptive Control (EHAC) and Generalized Predictive Control (GPC). In this work, among these number of MPC algorithms GPC is studied in detail.

Generalized Predictive Control (GPC)

The GPC method was proposed by Clarke et al [1] and has become one of the most popular MPC methods both in industry and academia. It has been successfully implemented in many industrial applications, showing good performance and a certain degree of robustness.

The basic idea of GPC is to calculate a sequence of future control signals in such a way that it minimizes a multistage cost function defined over a prediction horizon. The index to be optimized is the expectation of a quadratic function measuring the distance between the predicted system output and some reference sequence over the horizon plus a quadratic function measuring the control effort.

Generalized Predictive Control has many ideas in common with the other predictive controllers since it is based upon the same concepts but it also has some differences. As will be seen later, it provides an analytical solution (in the absence of constraints), it can deal with unstable and non-minimum phase plants and incorporates the concept of control horizon as well as the consideration of weighting of control increments in the cost function. The general set of choices available for GPC leads to a greater variety of control objective compared to other approaches, some of which can be considered as subsets or limiting cases of GPC.

The GPC scheme can be seen in Fig. 1. It consists of the plant to be controlled, a reference model that specifies the desired performance of the plant, a linear model of the plant, and the Cost Function Minimization (CFM) algorithm that determines the input needed to produce the plant's desired performance. The GPC algorithm consists of the CFM block.

The GPC system starts with the input signal, *r(t),* which is presented to the reference model. This model produces a tracking reference signal, *w (t) that* is used as an input to the CFM block. The CFM algorithm produces an output, which is used as an input to the plant. Between samples, the CFM algorithm uses this model to calculate the next control input, *u (t+1),* from predictions of the response from the plant's model. Once the cost function is minimized, this input is passed to the plant. This algorithm is outlined below.

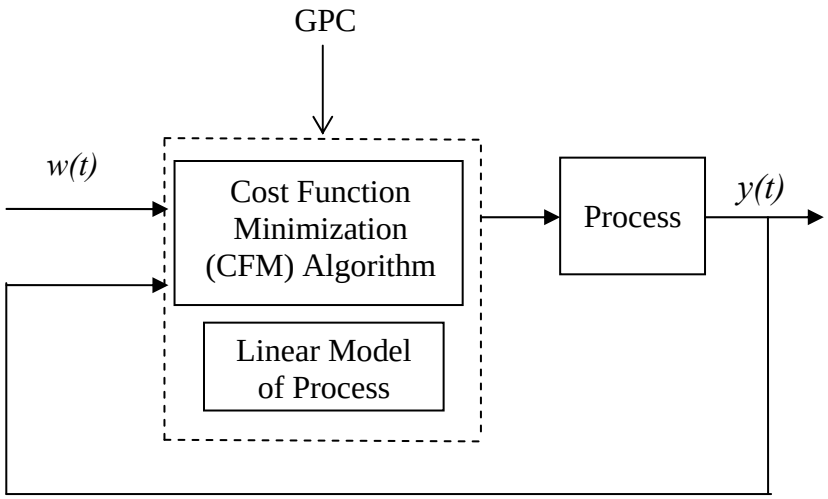


Figure 1. *Basic Structure of GPC*

Formulation of Generalized Predictive Control

Most single-input single-output (SISO) plants, when considering operation around particular set points and after linearization, can be described by Equation (1) [2].

$$A(z^{-1})y(t) = z^{-d}B(z^{-1})u(t-1) + C(z^{-1})e(t) \quad (1)$$

where $u(t)$ and $y(t)$ are the control and output sequence of the plant and $e(t)$ is a zero mean white noise. A, B And C are the following polynomials in the backward shift operator z^{-1}:

$$A(z^{-1}) = 1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_{na} z^{-na}$$

$$B(z^{-1}) = b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_{nb} z^{-nb}$$

$$C(z^{-1}) = 1 + c_1 z^{-1} + c_2 z^{-2} + ... + c_{nc} z^{-nc}$$

Where, d is the dead time of the system. This model is known as a Controller Auto-Regressive Moving-Average (CARIMA) model. It has been argued that for many industrial applications in which disturbances are non-stationary an integrated CARMA (CARIMA) model is more appropriate. A CARIMA model is given by Equation (2):

$$A(z^{-1})y(t) = z^{-d}B(z^{-1})u(t-1) + C(z^{-1})\frac{e(t)}{\Delta} \tag{2}$$

with $\Delta = 1 - z^{-1}$

For simplicity, C polynomial in Equation (2) is chosen to be 1. Notice that if C^{-1} can be truncated it can be absorbed into A and B.

Cost Function

GPC algorithm consists of applying a control sequence that minimizes a multistage cost function of the form given in Equation (3)

$$J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} \delta(j)\left[\hat{y}(t+j|t) - w(t+j)\right]^2 + \sum_{j=1}^{N_u} \lambda(j)\left[\Delta u(t+j-1)\right]^2 \tag{3}$$

where $\hat{y}(t+j\,|\,t)$ is an optimum j-step ahead prediction of the system output on data up to time *k*, N_1 and N_2 are the minimum and maximum costing horizons, N_u control horizon, $\delta(j)$ and $\lambda(j)$ are weighing sequences and $w(t+j)$ is the future reference trajectory, which can considered to be constant.

The objective of predictive control is to compute the future control sequence $u(t)$, $u(t+1), \ldots u(t+N_u)$ in such a way that the future plant output $y(t+j)$ is driven close to $w(t+j)$. This is accomplished by minimizing $J(N_1, N_2, N_u)$.

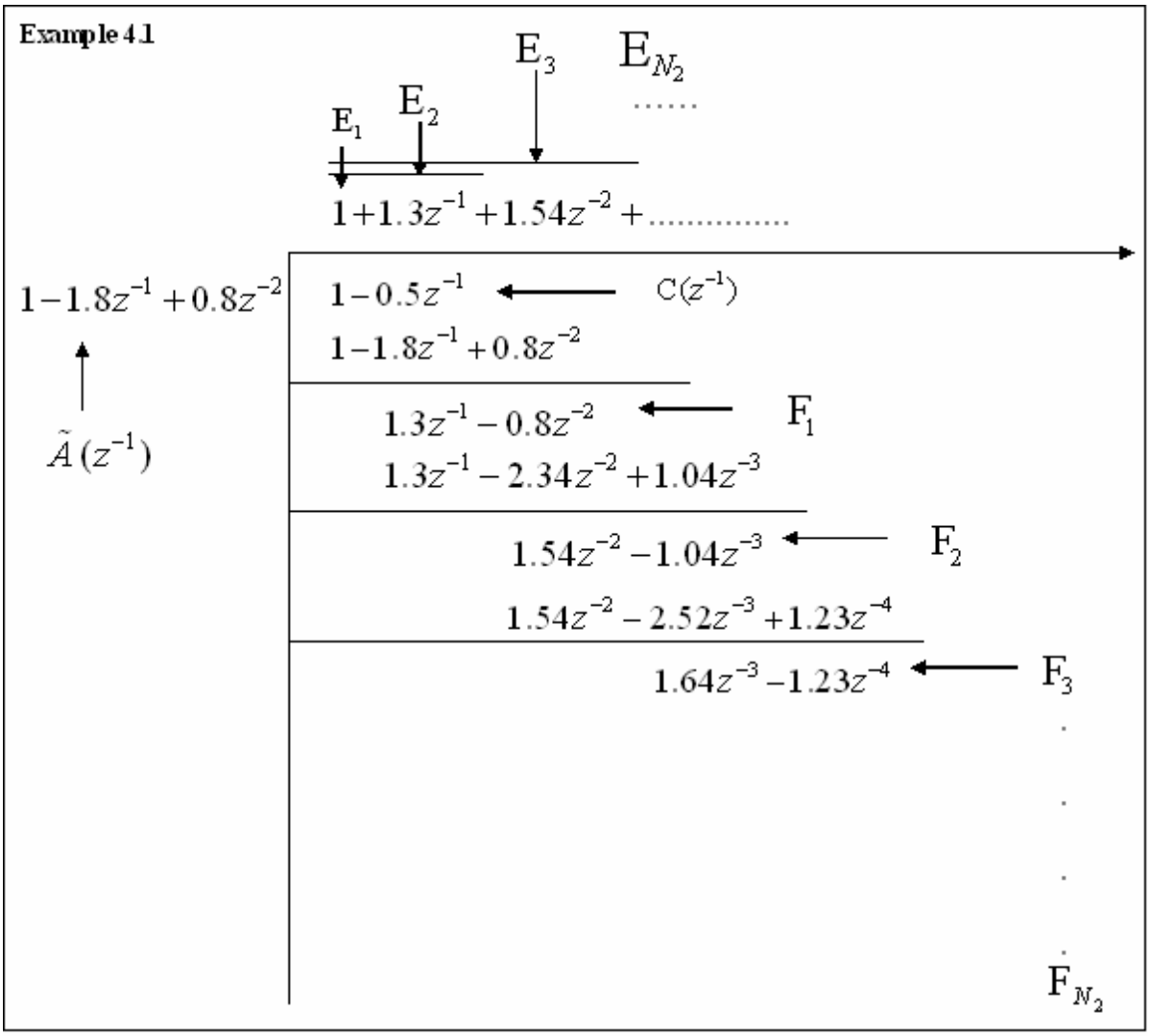
Cost Function Minimization Algorithm

In order to optimize the cost function the optimal prediction of $y(t+j)$ for $j \geq N_1$ and $j \leq N_2$ is required. To compute the predicted output, consider the following Diophantine Equation (4):

$$1 = E_j(z^{-1})\,\tilde{A}(z^{-1}) + z^{-j}F_j(z^{-1}) \text{ With } \tilde{A}(z^{-1}) = \Delta A(z^{-1}) \tag{4}$$

The polynomials E_j and F_j are uniquely defined with degrees $j-1$ and na respectively. They can be obtained dividing 1 by $\tilde{A}(z^{-1})$ until the remainder can be factorized

as $z^{-j}F_j(z^{-1})$. The quotient of the division is the polynomial $E_j(z^{-1})$. An example demonstrating calculation of E_j and Fj coefficients in Diophantine Equation is shown in Example 1 below:



***Example 1**: Diophantine Equation Demonstration Example*

Introduction to Neural Generalized Predictive Control

The ability of the GPC to make accurate predictions can be enhanced if a neural network is used to learn the dynamics of the plant instead of standard nonlinear modeling techniques. [3] The selection of the minimization algorithm affects the computational efficiency of the algorithm. Explicit solution for it can be obtained if the criterion is quadratic, the model is linear and there are no constraints; otherwise an iterative optimization method has to be used. In this project work Newton-Raphson method is used as the optimization

algorithm. The main cost of the Newton-Raphson algorithm is in the calculation of the Hessian, but even with this overhead the low iteration numbers make Newton-Raphson a faster algorithm for real-time control [4].

The Neural Generalized Predictive Control (NGPC) system can be seen in Fig. 2. It consists of four components, the plant to be controlled, a reference model that specifies the desired performance of the plant, a neural network that models the plant, and the Cost Function Minimization (CFM) algorithm that determines the input needed to produce the plant's desired performance. The NGPC algorithm consists of the CFM block and the neural net block.

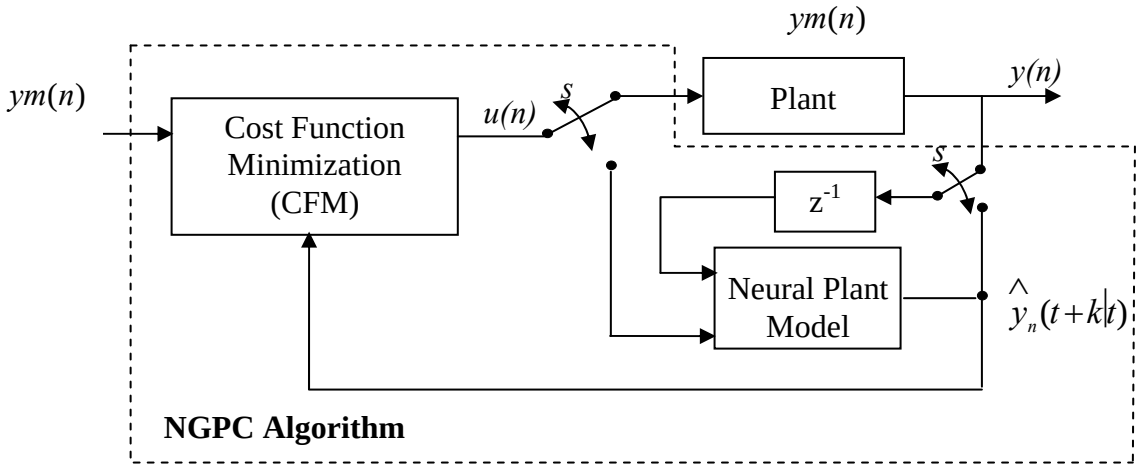


***Figure 2**. Block Diagram of NGPC System*

The NGPC system starts with the input signal, *r(n),* which is presented to the reference model. This model produces a tracking reference signal, *ym(n),* that is used as an input to the CFM block. The CFM algorithm produces an output that is either used as an input to the plant or the plant's model. The double pole double throw switch, **S,** is set to the plant when the CFM algorithm has solved for the best input, *u(n)*, that will minimize a specified cost function. Between samples, the switch is set to the plant's model where the CFM algorithm uses this model to calculate the next control input, *u(n+1)***,** from predictions of the response from the plant's model. Once the cost function is minimized, this input is passed to the plant. This algorithm is outlined below.

The computational performance of a GPC implementation is largely based on the minimization algorithm chosen for the CFM block. The selection of a minimization method can be based on several criteria such as: number of iterations to a solution, computational costs and accuracy of the solution. In general these approaches are iteration intensive thus

making real-time control difficult. In this work Newton-Raphson as an optimization technique is used. Newton-Raphson is a quadratic ally converging. The improved convergence rate of Newton-Raphson is computationally costly, but is justified by the high convergence rate of Newton-Raphson.

The quality of the plant's model affects the accuracy of a prediction. A reasonable model of the plant is required to implement GPC. With a linear plant there are tools and techniques available to make modeling easier, but when the plant is nonlinear this task is more difficult. Currently there are two techniques used to model nonlinear plants. One is to linearize the plant about a set of operating points. If the plant is highly nonlinear the set of operating points can be very large. The second technique involves developing a nonlinear model which depends on making assumptions about the dynamics of the nonlinear plant. If these assumptions are incorrect the accuracy of the model will be reduced.

Models using neural networks have been shown to have the capability to capture nonlinear dynamics**.** For nonlinear plants, the ability of the GPC to make accurate predictions can be enhanced if a neural network is used to learn the dynamics of the plant instead of standard modeling techniques. Improved predictions affect rise time, over-shoot, and the energy content of the control signal.

Formulation of NGPC

Cost Function

As mentioned earlier, the NGPC algorithm [4] is based on minimizing a cost function over a finite prediction horizon. The cost function of interest to this application is

$$J(N_1, N_2, N_u) = \sum_{j=N_1}^{N_2} \delta(j)\left[\hat{y}_n(t+j|t) - w(t+j)\right]^2 + \sum_{j=1}^{N_u} \lambda(j)\left[\Delta u(t+j-1)\right]^2 \quad (5)$$

N_1 = Minimum Costing Prediction Horizon
N_2 = Maximum Costing Prediction Horizon
N_u= Length of Control Horizon
$\hat{y}(t+k|t)$ = Predicted Output from Neural; Network
$u(t+k|t)$ = Manipulated Input
$w(t+k)$ = Reference Trajectory
δ *and* λ = Weighing Factor

This cost function minimizes not only the mean squared error between the reference signal and the plant's model, but also the weighted squared rate of change of the control input with it's constraints. When this cost function is minimized, a control input that meets the constraints is generated that allows the plant to track the reference trajectory within some tolerance. There are four tuning parameters in the cost function, N_1, N_2, N_u, and λ. The predictions of the plant will run from N_1 to N_2 future time steps. The bound on the control horizon is N_u. The only constraint on the values of Nu and N_1 is that these bounds must be less than or equal to N_2. The second summation contains a weighting factor, λ that is introduced to control the balance between the first two summations. The weighting factor acts as a damper on the predicted $u(n+1)$.

Cost Function Minimization Algorithm

The objective of the CFM algorithm is to minimize J in Equation (6) with respect to $[u(n+1), u(n+2), \ldots, u(n+N_u)]^T$, denoted U. This is accomplished by setting the Jacobian of Equation (5) to zero and solving for U. With Newton-Rhapson used as the CFM algorithm, J is minimized iteratively to determine the best U. An iterative process yields intermediate values for J denoted $J(k)$. For each iteration of $J(k)$ an intermediate control input vector is also generated and is denoted as in Equation (6):

$$U(k) \overset{\Delta}{=} \begin{bmatrix} u(t+1) \\ u(t+2) \\ \cdot \\ \cdot \\ \cdot \\ u(t+N_u) \end{bmatrix} \qquad \text{k=1,}\ldots\ldots N_u \qquad (6)$$

Newton-Raphson method is one of the most widely used of all root-locating formula. If the initial guess at the root is x_i, a tangent can be extended from the point $[x_i, f(x_i)]$. The point where this tangent crosses the x-axis usually represents an improved estimate of the root. So the first derivative at x on rearranging can be given as: $x_{(i+1)} = x_{(i)} - \dfrac{f(x_i)}{f'(x_i)}$

Using this Newton-Raphson update rule, $U(k+1)$ is given by Equation (7)

$$U(k+1) = U(k) - \left(\frac{\partial^2 J}{\partial U^2}(k) \right)^{-1} \frac{\partial J}{\partial U}(k)'$$

Where $$f(x) = \frac{\partial J}{\partial U} \tag{7}$$

And the Jacobian is denoted as in Equation (8)

$$\frac{\partial J}{\partial U}(k) \equiv \begin{bmatrix} \frac{\partial J}{\partial u(t+1)} \\ . \\ . \\ . \\ \frac{\partial J}{\partial u(t+N_u)} \end{bmatrix} \tag{8}$$

And the Hessian as in Equation (9)

$$\frac{\partial^2 J}{\partial U^2}(k) \equiv \begin{bmatrix} \frac{\partial^2 J}{\partial u(t+1)^2} & . & . & \frac{\partial^2 J}{\partial u(t+1)\partial u(t+N_u)} \\ . & . & . & . \\ . & . & . & . \\ \frac{\partial^2 J}{\partial u(t+N_u)\partial u_k(t+1)} & . & . & \frac{\partial^2 J}{\partial u(t+N_u)^2} \end{bmatrix} \tag{9}$$

Each element of the Jacobianis calculated by partially differentiating Equation (8) with respect to vector U.

$$\frac{\partial J}{\partial u(t+h)} = 2\sum_{j=N_1}^{N_2} \delta(j) \left[\hat{y}_n(t+j) - w(t+j) \right] \frac{\partial \hat{y}_n(t+j)}{\partial u(t+h)} + 2\sum_{j=1}^{N_u} \lambda(j) \left[\Delta u(t+j) \right] \frac{\partial \Delta u(t+j)}{\partial u(t+h)} \tag{10}$$

Where, $h = 1, ..., N_u$

Once again Equation (10) is partially differentiated with respect to vector U to get each element of the Hessian.

$$\frac{\partial^2 J}{\partial u(t+m)\partial u(t+h)} = 2\sum_{j=N_1}^{N_2} \delta(j) \left\{ \frac{\partial^2 \hat{y}(t+j)}{\partial u(t+m)\partial u(t+h)} \left[\hat{y}(t+j) - w(t+j) \right] - \frac{\partial \hat{y}(t+j)}{\partial u(t+m)} \frac{\partial \hat{y}(t+j)}{\partial u(t+h)} \right\}$$

$$+2\sum_{j=N_1}^{N_2} \lambda(j) \left\{ \frac{\partial \Delta n(t+j)}{\partial u(t+m)} \frac{\partial \Delta n(t+j)}{\partial u(t+h)} + \frac{\partial^2 \Delta n(t+j)}{\partial u(t+m)\partial u(t+h)} \right\} \tag{11}$$

The m^{th}, h^{th} elements of the Hessian matrix in Equation (11) are $h = 1, ..., N_u$ and $m = 1, ..., N_u$.

The last computation needed to evaluate $U(k+1)$ is the calculation of the predicted output of the plant, $\hat{y}(t+j)$, and it's derivatives. The next sections define the equation of a multilayer feed forward neural network, and define the derivative equations of the neural network.

Neural Network Architecture

In NGPC the model of the plant is a neural network. This neural model is constructed and trained using MATLAB Neural Network System Identification Toolbox commands [5].

The output of trained neural network is used as the predicted output of the plant. This predicted output is used in the Cost Function Minimization Algorithm. If $y_n(t)$ is the neural network's output then it is nothing but plant's predicted output $\hat{y}_n(t+k|t)$.

The initial training of the neural network is typically done off-line before control is attempted. The block configuration for training a neural network to model the plant is shown in Fig. 3. The network and the plant receive the same input, *u(t).* The network has an additional input that either comes from the output of the plant, *y(t),* or the neural network's, $y_n(t)$. The one that is selected depends on the plant and the application. This input assists the network with capturing the plant's dynamics and stabilization of unstable systems. To train the network, its weights are adjusted such that a set of inputs produces the desired set of outputs. An error is formed between the responses of the network, $y_n(t)$, and the plant, *y(t).* This error is then used to update the weights of the network through gradient descent learning. In this work a Levenberg-Marquardt method is used as gradient descent learning algorithm for updating the weights. This is standard method for minimization of mean-square error criteria, due to its rapid convergence properties and robustness. This process is repeated until the error is reduced to an acceptable level.

Since a neural network will be used to model the plant, the configuration of the network architecture should be considered. This implementation of NGPC adopts input/output models.

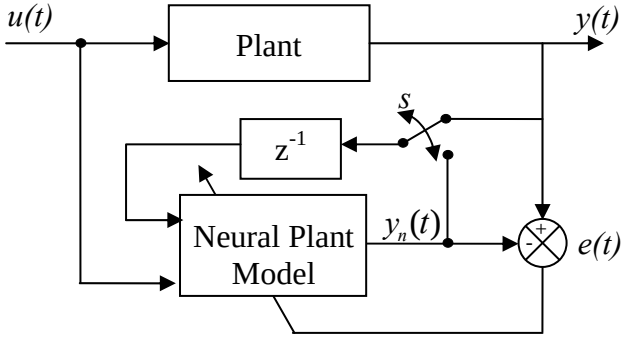


***Figure 3**. Block Diagram of Off-line Neural Network Training*

The diagram below, Fig.4**,** depicts a multi-layer feed-forward neural network with a time-delayed structure. For this example, the inputs to this network consists of two external inputs, *u(t)* and two outputs *y(t-1),* with their corresponding delay nodes, *u(t), u(t-1)* and *y(t-1), y(t-2)*. The network has one hidden layer containing five hidden nodes that uses bi-polar sigmoidal activation output function. There is a single output node which uses a linear output function, of one for scaling the output.

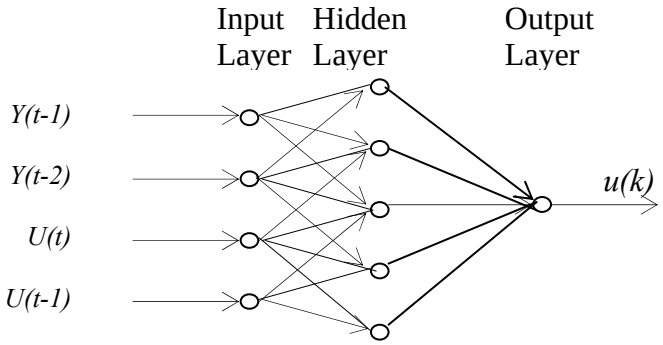


***Figure 4.** Neural Network Architecture*

The equation for this network architecture is:

$$y_n(t) = \sum_{j=1}^{hid} w_j f_j\left(net(t)\right)$$

And

$$net_j(t) = \sum_{j=1}^{n_d} w_{j,i+1} u(t-i) + \sum_{j=1}^{d_d} w_{j,n_d+i+1} y(t-i) \qquad (12)$$

Where,

$y_n(t)$ is the output of the neural network

$f_j(.)$	is the output function for the j^{th} node of the hidden layer
$net_j(t)$	is the activation level of the j^{th} node's output function
hid	is the number of hidden nodes in the hidden layer
n_d	is the number of input nodes associated with $u(.)$
d_d	is the number of input nodes associated with $y(.)$
w_j	is the weight connecting the j^{th} hidden node to the output node
$w_{j,i}$	is the weight connecting the i^{th} hidden input node to the j^{th} hidden node
$y(t-i)$	is the delayed output of the plant used as input to the network
$u(t-i)$	is the input to the network and its delays

This neural network is trained in offline condition with plants input/output data

Prediction Using Neural Network

The NGPC algorithm uses the output of the plant's model to predict the plant's dynamics to an arbitrary input from the current time, *t,* to some future time, *t+k.* This is accomplished by time shifting equations Equation (11) and (12), by *k,* resulting in Equation (13) and (14).

$$y_n(t+k) = \sum_{j=1}^{hid} w_j f_j\left(net_j(t+k)\right) \tag{13}$$

And

$$net_j(t+k) = \sum_{i=1}^{hid} w_{j,i+1} \begin{cases} u(n+k-i), k-N_u < i \\ u(n+N_u), k-N_u \geq i \end{cases} + \sum_{j=1}^{\min(k,d_d)} \left(w_{j,n_d+i+1} y_n(t+k-i)\right) \tag{14}$$

$$+ \sum_{i=k+1}^{d_d} \left(w_{j,n_d+i+1} y(t+k-i)\right)$$

The first summation of Equation (14) breaks the input into two parts represented by the conditional. The condition where $k-N_u < i$ handles the previous future values of the u up to $u(t+N_u-1)$. The condition where $k-N_u > i$ sets the input from $u(t+N_u)$ to $u(t+k)$ equal to $u(t+N_u)$. The second condition will only occur if $N_2 > N_u$. The next summation of Equation (14) handles the recursive part of prediction. This feeds back the network output, yn, for k or d_d times, which ever is smaller. The last summation of Equation (14)

handles the previous values of *y*. The following section derives the derivatives of Equation (13) and (14) with respect to the input $u(t+h)$.

Neural Network Derivative Equations

To evaluate the Jacobian and the Hessian in Equation (8) and (9) the network's first and second derivative with respect to the control input vector are needed.

Jacobian Element Calculation

The elements of the Jacobian are obtained by differentiating $y_n(t+k)$ in Equation (13) with respect to $u(t+h)$ resulting in

$$\frac{\partial yn(t+k)}{\partial u(t+h)} = \sum_{j=1}^{hid} w_j \frac{\partial f_j\left(net_j(t+k)\right)}{\partial u(t+h)} \tag{15}$$

Applying chain rule to $\partial f_j\left(net_j(t+k)\right)/\partial u(t+h)$ results in

$$\frac{\partial f_j\left(net_j(t+k)\right)}{\partial u(t+h)} = \frac{\partial f_j\left(net_j(t+k)\right)}{\partial net_j(t+k)} \frac{\partial net_j(t+k)}{\partial u(t+h)} \tag{16}$$

Where $\partial f_j\left(net_j(t+k)\right)/\partial net_j(t+k)$ is the output function's derivative which will become zero as we are using a linear (constant value) output activation function and

$$\frac{\partial net_j(t+k)}{\partial u(t+h)} = \sum_{i=0}^{n_d} w_{j,i+1} \begin{cases} \delta(k-i,h), k-N_u < i \\ \delta(N_u,h), k-N_u \geq i \end{cases} + \sum_{i=0}^{\min(k,d_d)} w_{j,i+n_d+1} \frac{\partial yn(t+k-i)}{\partial u(t+h)} \delta_1(k-i-1) \tag{17}$$

Note that in the last summation of Equation (17) the step function, δ**,** was introduced. This was added to point out that this summation evaluates to zero for $k-i<l$, thus the partial does not need to be calculated for this condition.

Hessian Element Calculation

Hessian elements are obtained by once again differentiating Equations (15) by *u(t+m),* resulting in Equation (18):

$$\frac{\partial^2 yn(t+k)}{\partial u(t+h)\partial u(t+m)} = \sum_{i=0}^{n_d} w_j \frac{\partial^2 f_j\left(net_j(t+k)\right)}{\partial u(t+h)\partial u(t+m)} \tag{18}$$

Where,

$$\frac{\partial^2 f_j\left(net_j(t+k)\right)}{\partial u(t+h)\partial u(t+m)} = \frac{\partial f_j\left(net_j(t+k)\right)}{\partial net_j(t+k)} \frac{\partial^2 net_j(t+k)}{\partial u(t+h)\partial u(t+m)} + \frac{\partial^2 f_j\left(net_j(t+k)\right)}{\partial net_j(t+k)^2} \frac{\partial net_j(t+k)}{\partial u(t+h)} \frac{\partial net_j(t+k)}{\partial u(t+m)} \tag{19}$$

Equation (19) is the result of applying the chain rule twice

Simulation Results

The objective of this study is to show how GPC and NGPC implementation can cope with linear systems. GPC is applied to the systems with changes in system order. The Neural based GPC is implemented using MATLAB Neural Network Based System Design Toolbox. [5].

GPC and NGPC for Linear Systems

The above derived GPC and NGPC algorithm is applied to the different linear models with varying system order, to test its capability. Carrying out simulation in MATLAB 7.0.1 does this. Different systems with large dynamic differences are considered for simulation. GPC and NGPC are showing robust performance for these systems. In below figures, for every individual system the systems output with GPC and NGPC is plotted in single figure for comparison purpose. Also the control efforts taken by the both controllers are plotted in consequent figures for every individual figure.

In this simulation, neural network architecture considered is as follows. The inputs to this network consists of two external inputs, *u(t)* and two outputs *y(t-1),* with their corresponding delay nodes, *u(t), u(t-1)* and *y(t-1), y(t-2)*. The network has one hidden layer containing five hidden nodes that uses bi-polar sigmoid activation output function. There is a single output node that uses a linear output function, of one for scaling the output.

For all the systems Prediction Horicon N_1 =1, N_2 =7 and Control Horizon (N_u) is 2. The weighing factor λ for control signal is kept to 0.3 and δ for reference trajectory is set to 0. The same controller setting is used for all the systems simulation. The following simulation results are obtained showing the Plant output when GPC and NGPC are applied. Also the required Control action for different systems is shown.

System I**:** The GPC and NGPC algorithms are applied to a second order system given in Equation (20). Fig. 5 shows the plant output when GPC and NGPC. Fig. 6 shows the control efforts taken by both controllers.

$$G(s) = \frac{1}{1+10s+40s^2} \tag{20}$$

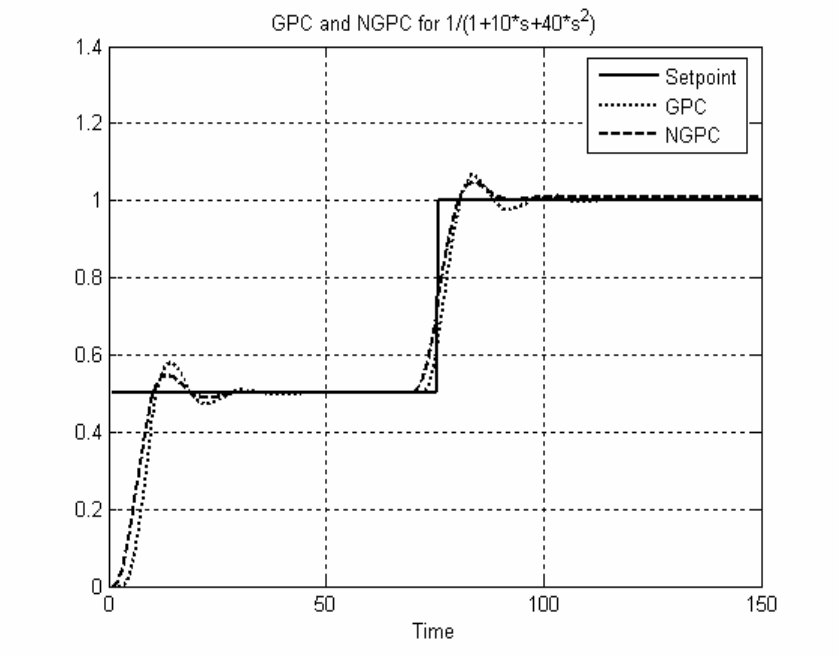


Figure 5. *System I Output using GPC and NGPC*

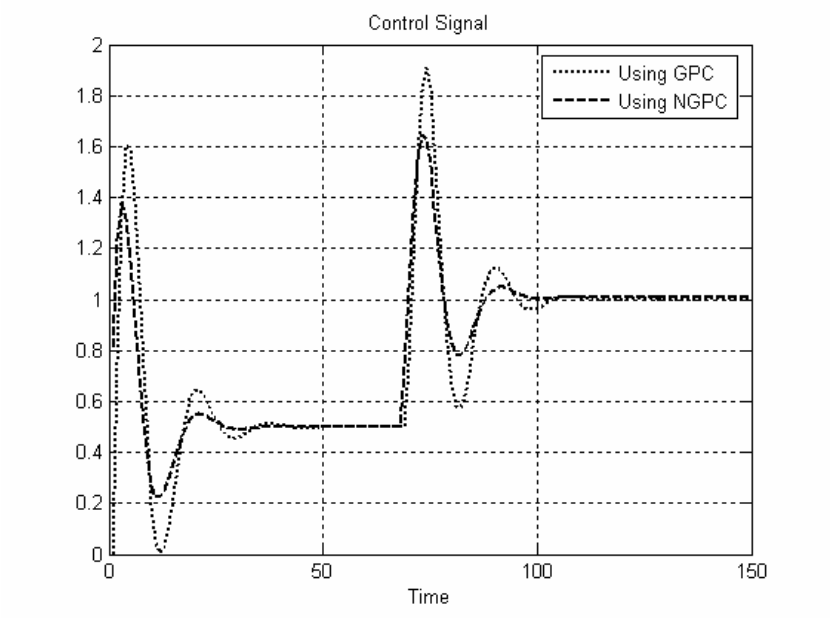


***Figure 6**. Control Signal for System I*

System II A simple first order system given by Equation (21) is controlled. Fig. 7 and Fig. 8 show the system output and control signal.

$$G(s) = \frac{1}{1+10s} \tag{21}$$

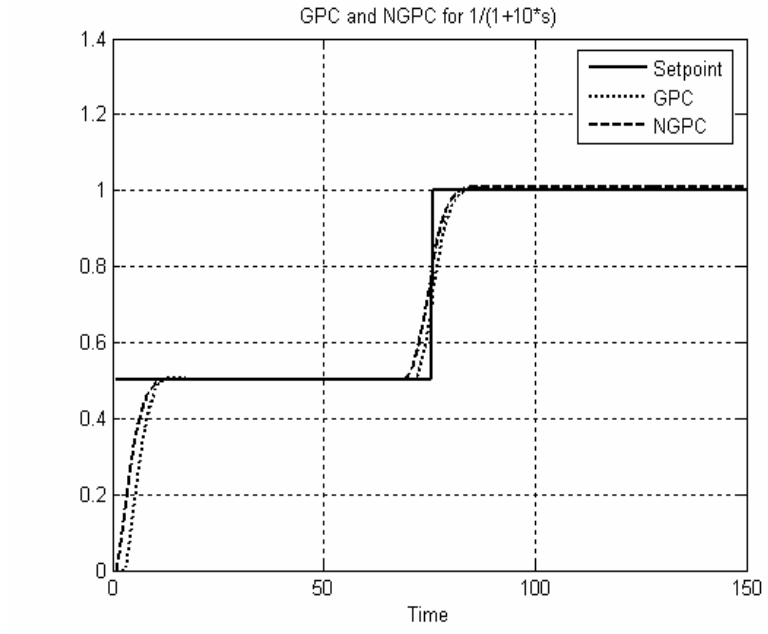


***Figure 7**. System II Output using GPC and NGPC*

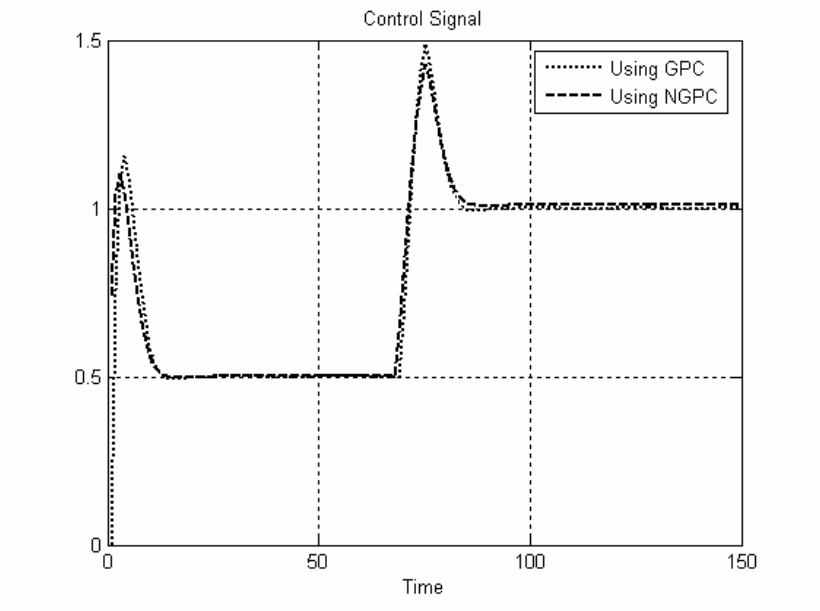


***Figure 8**. Control Signal for System II*

System III**:** A second order system given by Equation (22) is controlled using GPC. Fig. 9 and Fig10show the predicted output and control signal.

$$G(s) = \frac{1}{10s(1+2.5s)} \tag{22}$$

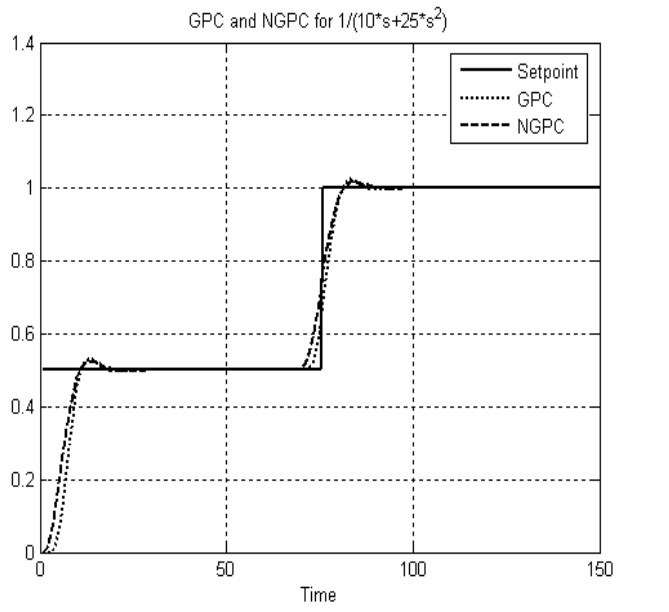


***Figure 9**. System III Output using GPC and NGPC*

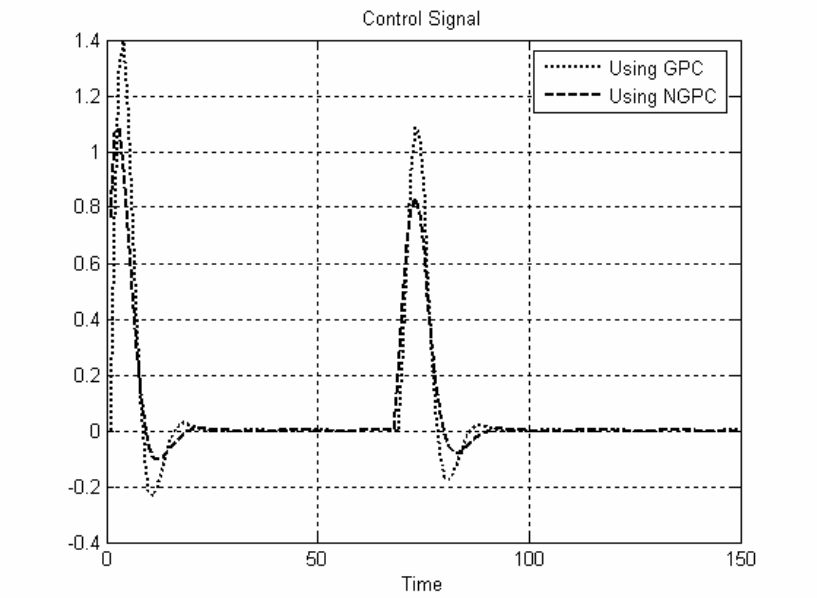


Figure 10. *Control Signal for System III*

Before applying NGPC to the all above systems it is initially trained using Levenberg-Marquardt learning algorithm. Fig. 10(a) shows Input data applied to the neural network for offline training purpose. Fig. 10(b) shows the corresponding neural network output.

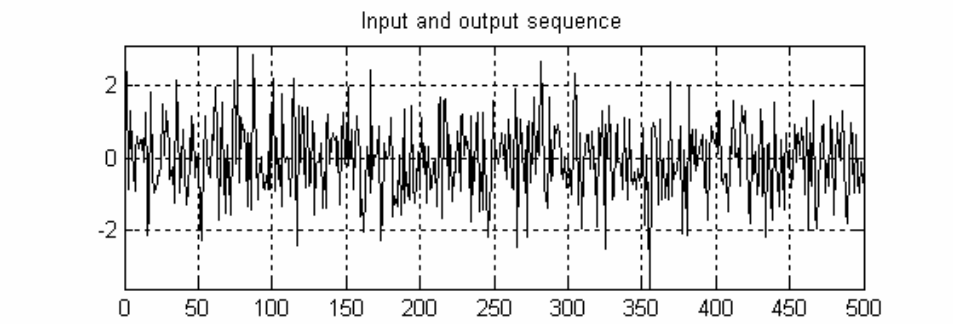


Figure 10 *(a). Input Data for Neural Network*

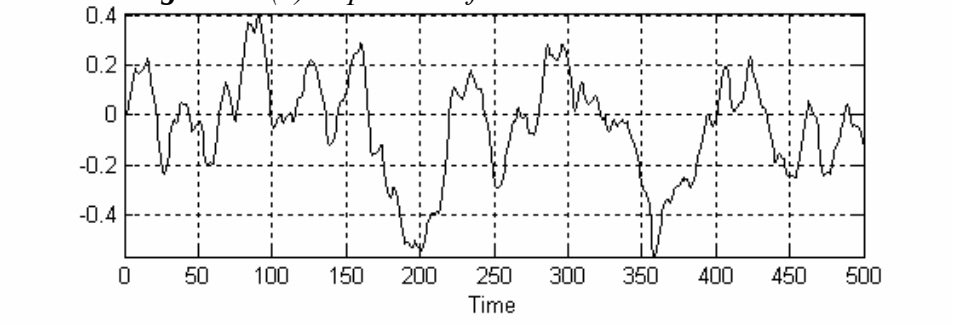


Figure 10 (b*). Neural Network Response for Random*

To check whether this neural network is trained to replicate it as a perfect model or not, common input is applied to the trained neural network and plant. Fig. 11(a) shows the

trained neural networks output & predicted output for common input. Also the error between these two responses is shown in Fig. 11(b).

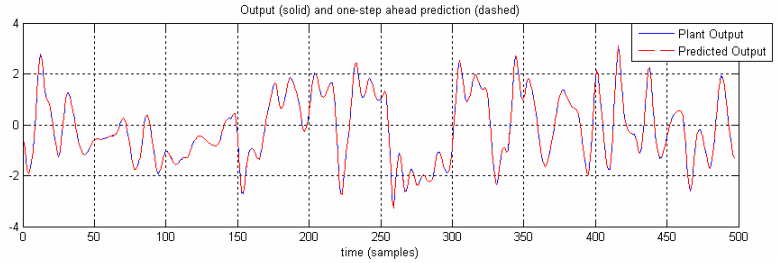


Figure 11(a). Neural Network & Plant Output

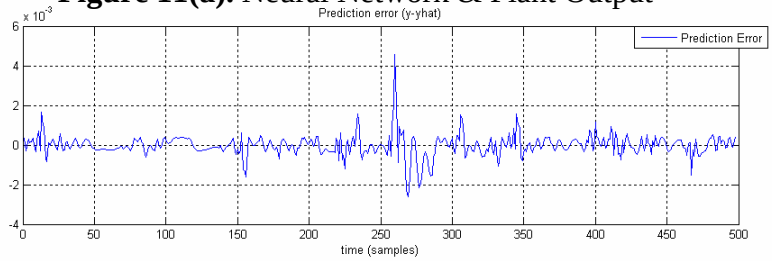


Figure 11(b). Error between Neural Network & Plant Output

Performance evaluation of both the controller is carried out using ISE and IAE criteria given by the following equations:

$$ISE = \int_0^t e^2 dt$$

$$IAE = \int_0^t |e| dt$$

Table 1 gives ISE and IAE values for both GPC and NGPC implementation for all the linear systems given by Equation 20 to Equation 21. We can find that for each system ISE and IAE for NGPC is smaller or equal to GPC. So using GPC with neural network i.e. NGPC control configuration for linear application, is also a better choice.

Table 5.1: ISE and IAE Performance Comparison of GPC and NGPC for Linear System

Systems	Setpoint	GPC		NGPC	
		ISE	IAE	ISE	IAE
System I	0.5	1.6055	4.4107	1.827	3.6351
	1	0.2567	1.4492	0.1186	1.4312
System II	0.5	1.1803	3.217	0.7896	2.6894
	1	0.1311	0.767	0.063	1.017
System III	0.5	1.4639	3.7625	1.1021	3.3424
	1	0.1759	0.9065	0.0957	0.7062
	1	0.1311	0.767	0.063	1.017

Conclusion

In this paper a conventional Generalized Predictive Control algorithm is derived in detail. The capability of the algorithm is tested on variety of systems. An efficient implementation of GPC using a multi-layer feed-forward neural network as the plant's nonlinear model is presented to extend the capability of GPC i.e. NGPC for controlling linear process very efficiently.

References

1. D. W. Clarke, C. Mohtadi and P. S. Tuffs, *Generalized predictive control-part I. the basic algorithm*, Automatica, vol 23, pp. 137-148. 1987

2. Jacek M. Zurada, *Introduction to Artificial Neural Systems,* Jaico Publishing House, 2006.

3. Xue-Mei Sun, Chang-Ming Ren, Pi-Lian He, Yu-Hong Fan, *Predictive control based on neural network for nonlinear system with time delays*, IEEE proceeding , , pp. 319-322. 2002

4. Donald Soloway, *Neural generalized predictive control*, Proceeding IEEE International Symposium on Intelligent Control, Dearborn, , pp. 277-282. 1996

5. M. Nørgaard, *Neural Network Based Control System Design Toolkit*, ver.2 Tech. Report. 00-E-892, Department of Automation, Technical University of Denmark, 2000**.**

6. E. P. Nahas, M. A. Henson, D. E. Seborg, *Nonlinear internal model control strategy for neural network models*, Computers Chemical Engineering, vol.16, pp. 1039-1057. 1992.