Development of Explicit Neural Predictive Control Algorithm Using Particle Swarm Optimisation

Maciej Ławryńczuk

Institute of Control and Computation Engineering, Warsaw University of Technology
ul. Nowowiejska 15/19, 00-665 Warsaw, Poland
M.Lawrynczuk@ia.pw.edu.pl

Abstract. This paper describes development of a nonlinear Model Predictive Control (MPC) algorithm. The algorithm is very computationally efficient because for control signal calculation an explicit control law is used, no on-line optimisation is necessary. The control law is implemented by a neural network which is trained off-line by means of a particle swarm optimisation algorithm. Inefficiency of a classical gradient-based training algorithm is demonstrated for the polymerisation reactor. Moreover, the discussed MPC algorithm is compared in terms of accuracy and computational complexity with two suboptimal MPC algorithms with model linearisation and MPC with full nonlinear optimisation.

Keywords: Process control, Model Predictive Control, neural networks, optimisation, particle swarm optimisation, soft computing.

1 Introduction

Model Predictive Control (MPC) is a computer control strategy in which the control action is optimised over some future time horizon [8,15]. Thanks to the fact that a dynamic model is used for prediction of the future behaviour of the process, MPC algorithms, unlike any other control technique, can easily take into account constraints imposed on process inputs (manipulated variables) and outputs (controlled variables), which usually decide on quality, economic efficiency and safety. Secondly, MPC can be efficiently used for multivariable processes, with many inputs and outputs. As a result, MPC algorithms have been successfully used for years in many areas [14].

Because behaviour of numerous processes is typically nonlinear, nonlinear models, rather than simple linear ones, are used for prediction in MPC [3,10,15]. Although different types of nonlinear models can be used in MPC, neural models are particularly interesting. In order to reduce complexity of on-line calculations, suboptimal MPC algorithms are more and more popular in which the neural model is successively linearised on-line and the obtained linear approximation is used for prediction. Thanks to linearisation, the control signal can be calculated on-line from an easy to solve quadratic programming task [5,7,11,15]. To further

L. Rutkowski et al. (Eds.): ICAISC 2013, Part I, LNAI 7894, pp. 130–139, 2013.

reduce computational burden an explicit variant of the suboptimal MPC algorithm can be used in which constraints are treated somehow heuristically, but it makes it possible to replace quadratic programming with an explicit control law. The coefficients of this law are calculated on-line from a simple matrix decomposition task and a solution of a set of linear equations [6]. The necessity of model linearisation and matrix inversion can be also eliminated as shown in [4], in such a case the explicit control law is implemented by a neural network trained off-line. Data sets necessary for training and validation of such a neural network are generated by the classical explicit MPC algorithm. Unfortunately, development of the classical algorithm is an essential part of the design procedure, which may be a disadvantage.

In this paper alternative development of the explicit neural MPC algorithm is discussed. Unlike the algorithm presented in [4], the neural network is not trained to approximate behaviour of the classical explicit MPC algorithm, but the network used for control law calculation is trained directly off-line. Because such an optimisation problem may be difficult, non-convex and multimodal, a particle swarm optimisation algorithm is used. Efficiency of the discussed approach is demonstrated for the polymerisation process. Particle swarm optimisation approaches have been extensively used for global optimisation [1]. In control system engineering they have been used for off-line tuning parameters of the PID controller [12] and for on-line nonlinear optimisation in MPC algorithms [13,16].

2 Model Predictive Control Algorithms

In MPC algorithms [8,15] at each consecutive sampling instant k, $k = 0, 1, 2, \ldots$, a set of future control increments is calculated

$$\triangle \boldsymbol{u}(k) = [\triangle u(k|k)\ \triangle u(k+1|k) \ldots \triangle u(k+N_{\mathrm{u}}-1|k)]^{\mathrm{T}} \tag{1}$$

It is assumed that $\triangle u(k+p|k) = 0$ for $p \geq N_{\mathrm{u}}$, where N_{u} is the control horizon. The objective of the algorithm is to minimise differences between the reference trajectory $y^{\mathrm{ref}}(k+p|k)$ and predictions $\hat{y}(k+p|k)$ over the prediction horizon $N \geq N_{\mathrm{u}}$, i.e. for $p = 1, \ldots, N$. Assuming that constraints are imposed on the value and the rate of change of the input variable, future control increments (1) are determined from the following MPC optimisation task

$$\begin{aligned} &\min_{\triangle \boldsymbol{u}(k)} \left\{ \sum_{p=1}^{N} (y^{\mathrm{ref}}(k+p|k) - \hat{y}(k+p|k))^2 + \lambda \sum_{p=0}^{N_{\mathrm{u}}-1} (\triangle u(k+p|k))^2 \right\} \\ &\text{subject to} \\ &u^{\min} \leq u(k+p|k) \leq u^{\max},\ p = 0, \ldots, N_{\mathrm{u}} - 1 \\ &-\triangle u^{\max} \leq \triangle u(k+p|k) \leq \triangle u^{\max},\ p = 0, \ldots, N_{\mathrm{u}} - 1 \end{aligned} \tag{2}$$

where $\lambda > 0$ is a weighting coefficient. Only the first element of the determined sequence (1) is applied to the process, i.e. $u(k) = \triangle u(k|k) + u(k-1)$. At the next sampling instant, $k+1$, the prediction is shifted one step forward and the whole procedure is repeated.

3 Explicit Neural MPC Algorithm Using Particle Swarm Optimisation

Let the dynamic neural model of the process be described by

$$y(k) = f(\boldsymbol{x}(k)) = f(u(k-\tau), \ldots, u(k-n_{\mathrm{B}}), y(k-1), \ldots, y(k-n_{\mathrm{A}})) \tag{3}$$

where integers n_{A}, n_{B}, τ define the order of dynamics, $\tau \leq n_{\mathrm{B}}$. In such a case predictions $\hat{y}(k+p|k)$ are nonlinear functions of the calculated policy (1) and the whole optimisation problem (2) is nonlinear, frequently non-convex. That is why suboptimal MPC algorithms are frequently used in which at each sampling instant on-line a linear approximation of the model (3) is calculated. Thanks to linearisation, the MPC optimisation task (2) becomes a quadratic programming problem.

3.1 The Explicit Control Law

If the constraints are removed from the MPC optimisation task (2), one has

$$\min_{\triangle \boldsymbol{u}(k)} \left\{ J(k) = \left\| \boldsymbol{y}^{\mathrm{ref}}(k) - \hat{\boldsymbol{y}}(k) \right\|^2 + \| \triangle \boldsymbol{u}(k) \|^2_{\boldsymbol{\Lambda}} \right\} \tag{4}$$

where

$$\boldsymbol{y}^{\mathrm{ref}}(k) = \left[y^{\mathrm{ref}}(k+1|k) \ldots y^{\mathrm{ref}}(k+N|k) \right]^{\mathrm{T}}$$
$$\hat{\boldsymbol{y}}(k) = \left[\hat{y}(k+1|k) \ldots \hat{y}(k+N|k) \right]^{\mathrm{T}}$$

are vectors of length N, $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda, \ldots, \lambda)$ is a matrix of dimensionality $N_{\mathrm{u}} \times N_{\mathrm{u}}$. It can be shown [5] that if the linear approximation of the neural model is used for prediction, the output predictions are linear functions of the future control sequence $\triangle \boldsymbol{u}(k)$

$$\hat{\boldsymbol{y}}(k) = \boldsymbol{G}(k) \triangle \boldsymbol{u}(k) + \boldsymbol{y}^0(k) \tag{5}$$

where the matrix $\boldsymbol{G}(k)$ of dimensionality $N \times N_{\mathrm{u}}$ contains step-response coefficients of the linearised model, the vector $\boldsymbol{y}^0(k) = \left[y^0(k+1|k) \ldots y^0(k+N|k) \right]^{\mathrm{T}}$ is the free trajectory which depends only on the past. Using the prediction equation (5), the optimisation problem of the explicit MPC algorithm (4) becomes

$$\min_{\triangle \boldsymbol{u}(k)} \left\{ J(k) = \left\| \boldsymbol{y}^{\mathrm{ref}}(k) - \boldsymbol{G}(k) \triangle \boldsymbol{u}(k) - \boldsymbol{y}^0(k) \right\|^2 + \| \triangle \boldsymbol{u}(k) \|^2_{\boldsymbol{\Lambda}} \right\}$$

Due to the fact that the minimised cost function $J(k)$ is quadratic, optimal control moves can be calculated analytically, without any optimisation. One obtains the explicit control law

$$\triangle \boldsymbol{u}(k) = \boldsymbol{K}(k)(\boldsymbol{y}^{\mathrm{ref}}(k) - \boldsymbol{y}^0(k)) \tag{6}$$

where

$$\boldsymbol{K}(k) = (\boldsymbol{G}^{\mathrm{T}}(k)\boldsymbol{G}(k) + \boldsymbol{\Lambda})^{-1} \boldsymbol{G}^{\mathrm{T}}(k) \tag{7}$$

is a matrix of dimensionality $N_{\mathrm{u}} \times N$.

At each sampling instant k of the classical explicit MPC algorithm the following steps are repeated on-line:

1. The linear approximation of the neural model for the current operating point is found [5].
2. Step response coefficients which comprise the matrix $\boldsymbol{G}(k)$ are calculated [5].
3. The nonlinear free trajectory $\boldsymbol{y}^0(k)$ is calculated using a neural model of the process [5].
4. The matrix $\boldsymbol{K}(k)$ is calculated from Eq. (7).
5. The future control increments $\triangle\boldsymbol{u}(k)$ are found from Eq. (6).
6. The first element of the obtained vector $\triangle\boldsymbol{u}(k)$ is applied to the process.
7. Iteration number is increased ($k := k + 1$), the algorithm goes to step 1.

The same neural model is used for linearisation and the free trajectory calculation. Matrix inversion in Eq. (7) is calculated in a numerical efficient way using the LU (Low-Upper) matrix decomposition with partial pivoting [6].

3.2 The Algorithm with Direct Calculation of the Matrix $\boldsymbol{K_1(k)}$

The explicit MPC algorithm discussed in the following part of the paper is much simpler than the rudimentary explicit algorithm described in the previous subsection. First, the first element of the vector $\triangle\boldsymbol{u}(k)$ (i.e. the quantity $\triangle u(k|k)$) is only calculated. In place of the control law (6) the formula

$$\triangle u(k|k) = \boldsymbol{K}_1(k)(\boldsymbol{y}^{\mathrm{ref}}(k) - \boldsymbol{y}^0(k)) \tag{8}$$

is used where $\boldsymbol{K}_1(k)$ is the first row of the matrix $\boldsymbol{K}(k)$. Secondly, the nonlinear model is not linearised on-line, step-response coefficients of the linearised model and the dynamic matrix $\boldsymbol{G}(k)$ are not calculated on-line, the inverse matrix $(\boldsymbol{G}^{\mathrm{T}}(k)\boldsymbol{G}(k) + \boldsymbol{\Lambda})^{-1}$ is not calculated on-line. The vector $\boldsymbol{K}_1(k) = [k_{1,1}(k) \ldots k_{1,N}(k)]^{\mathrm{T}}$ for the current operating point is directly calculated by a neural network which is called a neural approximator. The algorithm uses two neural networks: NN_1 is a dynamic model of the process, NN_2 is a neural approximator. At each sampling instant k of the algorithm the following steps are repeated on-line:

1. The nonlinear free trajectory $\boldsymbol{y}^0(k)$ is calculated using a neural model of the process (the network NN_1).
2. The vector $\boldsymbol{K}_1(k)$ is calculated using the neural approximator (the network NN_2).
3. The current control increment $\triangle u(k|k)$ is found from Eq. (8).
4. The obtained solution is projected onto the admissible set of constraints.
5. The obtained solution is applied to the process.
6. Iteration number is increased ($k := k + 1$), the algorithm goes to step 1.

Although the control laws (6) and (8) can be easily derived forgetting the constraints imposed on the manipulated variable in the general MPC optimisation task (2), the obtained value of the control signal may not satisfy real limitations of the actuator. That is why the following constraints imposed on the currently calculated control signal are taken into account

$$u^{\min} \leq u(k|k) \leq u^{\max}, \ -\triangle u^{\max} \leq \triangle u(k|k) \leq \triangle u^{\max}$$

The control increment $\triangle u(k|k)$ calculated from Eq. (8) is hence projected onto the admissible set of constraints

$$\begin{aligned}
&\text{if } \triangle u(k|k) < -\triangle u^{\max} \ \triangle u(k|k) = -\triangle u^{\max} \\
&\text{if } \triangle u(k|k) > \triangle u^{\max} \ \triangle u(k|k) = \triangle u^{\max} \\
&u(k|k) = \triangle u(k|k) + u(k-1) \\
&\text{if } u(k|k) < u^{\min} \ u(k|k) = u^{\min} \\
&\text{if } u(k|k) > u^{\max} \ u(k|k) = u^{\max} \\
&u(k) = u(k|k)
\end{aligned} \tag{9}$$

3.3 Training the Network NN_2 Using Particle Swarm Optimisation

Elements of the vector $\boldsymbol{K}_1(k)$, i.e. scalars $k_{1,p}(k)$ for $p = 1, \ldots, N$, are calculated on-line by the neural approximator–the network NN_2 for the current operating point of the process. The operating point is defined by control signals applied to the process at some previous sampling instants and measurements of the output signal for the current and some previous instants. The quantities $k_{1,p}$ are hence functions of the following arguments

$$k_{1,p} = g_p(u(k-1), \ldots, u(k-\tilde{n}_{\mathrm{B}}), y(k), \ldots, y(k-\tilde{n}_{\mathrm{A}}))$$

where integers $\tilde{n}_{\mathrm{A}}$ and $\tilde{n}_{\mathrm{B}}$ define the current operating point. In this study the MultiLayer Perceptron (MLP) network with one hidden layer and linear outputs [2] is used as the NN_2 network. It has $\tilde{n}_{\mathrm{A}} + \tilde{n}_{\mathrm{B}} + 1$ inputs. Outputs of the network are described by the following equation

$$k_{1,p}(k) = w^2_{p,0} + \sum_{i=1}^{K} w^2_{p,i} \varphi \left(w^1_{i,0} + \sum_{j=1}^{\tilde{n}_{\mathrm{B}}} w^1_{i,j} u(k-j) + \sum_{j=0}^{\tilde{n}_{\mathrm{A}}} w^1_{i,\tilde{n}_{\mathrm{B}}+j+1} y(k-j) \right) \tag{10}$$

where K is the number of hidden nodes, φ denotes the transfer function of the hidden units (e.g. $\varphi = \tanh$), weights of the first layer are denoted by $w^1_{i,j}$ for $i = 1, \ldots, K$, $j = 1, \ldots, \tilde{n}_{\mathrm{A}} + \tilde{n}_{\mathrm{B}} + 1$, biases of the first layer are denoted by $w^1_{i,0}$ for $i = 1, \ldots, K$, weights of the second layer are denoted by $w^2_{p,i}$ for $p = 1, \ldots, N$, $i = 1, \ldots, K$, biases of the second layer are denoted by $w^2_{p,0}$ for $p = 1, \ldots, N$.

The training procedure for the network NN_2 is as follows. First, a series of random changes of the reference trajectory is assumed. These changes comprise the training data set, the number of the training patterns is S. Similarly, the validation data set is generated. Next, parameters of the network, i.e. weights, are optimised through simulations of the explicit MPC algorithm for the assumed training changes of the reference trajectory. The optimisation problem is defined for S training patterns

$$\begin{aligned} &\min_{w^1_{i,j},\ w^2_{p,i}} \left\{ \mathrm{SSE} = \sum_{k=1}^{S} \left[(y^{\mathrm{ref}}(k) - y(k))^2 + \lambda (\triangle u(k|k))^2 \right] \right\} \\ &\text{subject to} \\ &\triangle u(k|k) = \boldsymbol{K}_1(k)(\boldsymbol{y}^{\mathrm{ref}}(k) - \boldsymbol{y}^0(k)) \\ &u^{\min} \le u(k|k) \le u^{\max} \\ &-\triangle u^{\max} \le \triangle u(k|k) \le \triangle u^{\max} \end{aligned} \tag{11}$$

where $y(k)$ denotes the output of the simulated process for consecutive sampling instants $k = 1, \ldots, S$, elements of the vector $\boldsymbol{K}_1(k)$ are calculated form Eq. (10). Satisfaction of inequality constraints is enforced by the projection procedure (9). The optimisation task (11) is nonlinear, it may be non-convex and multimodal. That is why classical, gradient-based optimisation algorithms are likely to terminate at local minima. A straightforward choice is to use global optimisation methods. In this study the particle swarm optimisation algorithm is used.

4 Simulation Results

The considered example process is a polymerisation reaction taking place in a jacketed continuous stirred tank reactor [9]. The reaction is the free-radical polymerisation of methyl methacrylate with azo-bis-isobutyronitrile as initiator and toluene as solvent. The output NAMW (Number Average Molecular Weight) is controlled by manipulating the inlet initiator flow rate F_{I}. The reactor exhibits significantly nonlinear behaviour. It cannot be controlled efficiently by classical MPC schemes based on constant linear models [5,7,9,15].

The fundamental model (a set of ordinary differential equations solved using the Runge-Kutta RK45 method) is used as the real process during simulations. At first, the dynamic neural model NN_1 of the MLP type is developed. It has the general structure

$$y(k) = f(u(k-2), y(k-1), y(k-2))$$

As input and output variables have different orders of magnitude, they are scaled as $u = 100(F_{\mathrm{I}} - F_{\mathrm{I}0})$, $y = 0.0001(\mathrm{NAMW} - \mathrm{NAMW}_0)$ where $F_{\mathrm{I}0} = 0.028328$, $\mathrm{NAMW}_0 = 20000$ correspond to the initial operating point. The sampling time is 1.8 min. The network has 6 hidden nodes with the $\varphi = \tanh$ transfer function. For training the BFGS (Broyden-Fletcher-Goldfarb-Shanno) optimisation algorithm is used. Model development is thoroughly discussed in [5].

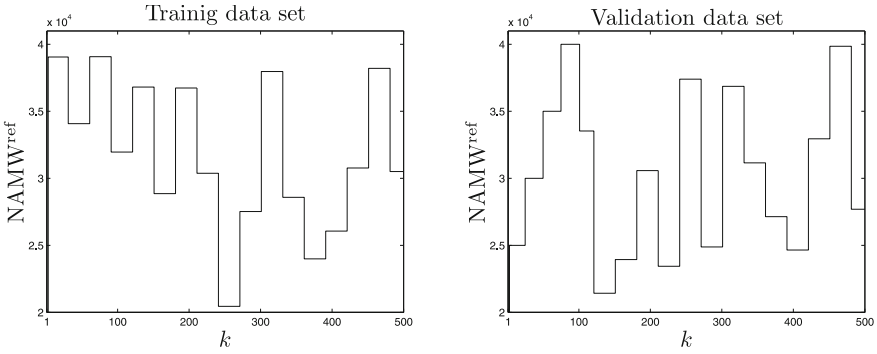

Fig. 1. The first 500 samples of the training data set (*left*) and the first 500 samples of the validation data set (*right*), the complete sets have 2000 samples

Table 1. Accuracy of the neural approximator NN_2 for gradient-based and particle swarm optimisation algorithms

Optimisation algorithm	$SSE_{training}$	$SSE_{validation}$
Gradient-based optimisation	5.273419×10^{11}	4.179207×10^{11}
Particle swarm optimisation	2.515997×10^{10}	3.319000×10^{10}

In order to train the neural approximator (the network NN_2), a series of random changes of the reference trajectory is generated. Both training and validation data sets have 2000 samples, the first quarters of them (for good presentation) are shown in Fig. 1. The prediction horizon is $N = 10$. The network has 2 inputs ($u(k-1)$ and $y(k)$), 3 hidden nodes with the $\varphi = \tanh$ transfer function and 9 outputs, due to process delay the quantity $k_{1,1}(k)$ is always 0. The optimisation problem (11) is solved by means of the gradient-based algorithm (the BFGS algorithm with shifted penalty function) and the particle swarm optimisation algorithm (the population size is 25, the maximum number of epochs is 2000). Numerical values of the obtained SSE (the Sum of Squared Errors) objective function are given in Table 1. The trajectories obtained as a result of optimisation in two compared algorithms much better demonstrate inefficiency of the classical approach and efficiency of the particle swarm optimisation algorithm. Fig. 2 shows the first quarters of input and output trajectories obtained in the gradient-based and particle swarm optimisation algorithms. In the first case the optimisation routine finds the solution which is a shallow local minima. Unfortunately, the explicit MPC does not follow the assumed reference trajectory. Conversely, the particle swarm optimisation algorithm finds parameters of the neural approximator NN_2 which gives good closed loop trajectories.

Next, the following four nonlinear MPC algorithms are compared:

a) the discussed explicit MPC algorithm with neural approximation and particle swarm optimisation used for off-line training of the network NN_2,

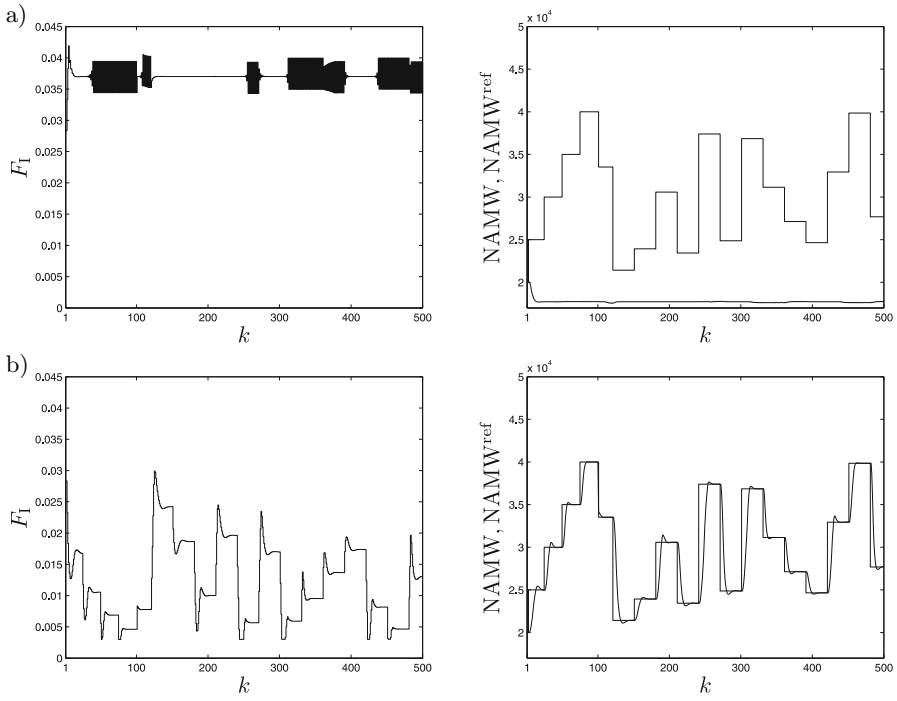

Fig. 2. The first 500 samples of input (F_{I}) and output (NAMW) trajectories for the validation data set trajectory NAMW$^{\mathrm{ref}}$ obtained in the explicit MPC algorithm: a) for optimisation of the neural approximator NN_2 the gradient algorithm is used, b) for optimisation the PSO algorithm is used

b) the classical explicit MPC algorithm with Nonlinear Prediction and Linearisation (MPC-NPL), with on-line successive linearisation of the neural dynamic model (the NN_1 network) and LU matrix decomposition [6],
c) the MPC-NPL algorithm with on-line successive linearisation of the neural dynamic model (the NN_1 network) and quadratic programming [5,7,15],
c) the MPC-NO algorithm with on-line nonlinear optimisation in which the neural dynamic model (the NN_1 network) is used for prediction without any simplifications [7,15].

Parameters of all MPC algorithms are the same $N = 10$, $\lambda = 0.2$, in the last three approaches $N_{\mathrm{u}} = 3$. The manipulated variable is constrained: $F_{\mathrm{I}}^{\min} = 0.003$, $F_{\mathrm{I}}^{\max} = 0.06$, $\triangle F_{\mathrm{I}}^{\max} = 0.005$. Fig. 3 shows trajectories obtained in the discussed explicit MPC algorithm and in the classical explicit MPC-NPL algorithm with on-line successive linearisation and LU matrix decomposition repeated at each sampling instant. Table 2 shows accuracy of all compared algorithms in terms of the SSE index and their computational complexity (in Millions of FLoating OperationS) for the whole simulation scenario (100 iterations). For the polymerisation process the all three suboptimal algorithms give control accuracy very close

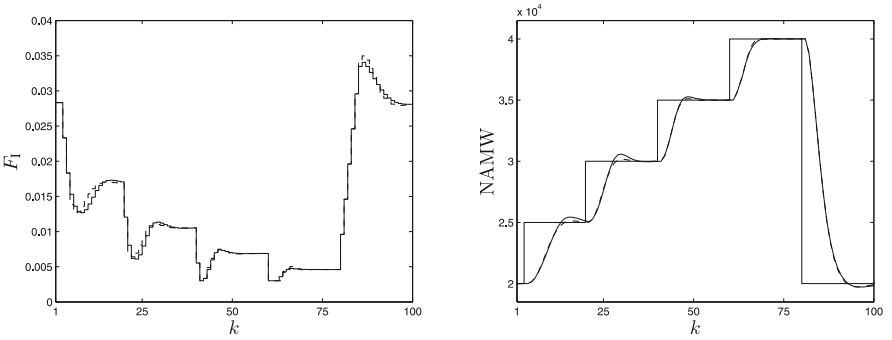

Fig. 3. Simulation results: the discussed explicit MPC algorithm (*solid line*) and the classical explicit MPC-NPL algorithm with on-line successive linearisation and LU matrix decomposition (*dashed line*)

Table 2. Accuracy (SSE) and computational load (MFLOPS) of compared nonlinear MPC algorithms

Algorithm	SSE	MFLOPS
Discussed explicit MPC with neural approximation	2.218638×10^9	0.164554
Explicit MPC-NPL with on-line LU decomposition	2.211703×10^9	0.217110
MPC-NPL with on-line quadratic programming	2.211703×10^9	0.404686
MPC-NO with on-line nonlinear optimisation	2.210627×10^9	4.109900

to that of the "ideal" computationally demanding MPC-NO approach. Moreover, the discussed explicit MPC algorithm works very similarly as the classical explicit MPC-NPL algorithm with on-line successive linearisation of the neural dynamic model and LU matrix decomposition. At the same time it is very computationally efficient: it is 25% more efficient when compared with the classical explicit algorithm and as much as 60% more efficient in comparison with the MPC-NPL algorithm with on-line quadratic programming.

5 Conclusions

The explicit MPC algorithm discussed in this paper is very computationally efficient because, unlike the classical explicit approach [6], successive on-line model linearisation and matrix calculations are not necessary. The current value of the control signal is calculated using a simple explicit formula and the neural approximator. Such a network can be trained off-line to mimic behaviour of the classical explicit MPC algorithm [4]. In this work an alternative development of the explicit neural MPC algorithm is discussed. The neural approximator is trained directly off-line, without the necessity of designing the classical explicit algorithm. As the resulting optimisation problem may be difficult, non-convex and multimodal, the particle swarm optimisation algorithm is used.

Acknowledgement. The work presented in this paper was supported by Polish national budget funds for science.

References

1. Eberhart, R.C., Shi, Y., Kennedy, J.: Swarm Intelligence. Morgan Kaufmann (2001)
2. Haykin, S.: Neural networks–a comprehensive foundation. Prentice Hall, Englewood Cliffs (1999)
3. Henson, M.A.: Nonlinear model predictive control: current status and future directions. Computers and Chemical Engineering 23, 187–202 (1998)
4. Ławryńczuk, M.: Explicit neural network-based nonlinear predictive control with low computational complexity. In: Szczuka, M., Kryszkiewicz, M., Ramanna, S., Jensen, R., Hu, Q. (eds.) RSCTC 2010. LNCS (LNAI), vol. 6086, pp. 649–658. Springer, Heidelberg (2010)
5. Ławryńczuk, M.: Neural networks in model predictive control. In: Nguyen, N.T., Szczerbicki, E. (eds.) Intelligent Systems for Knowledge Management. SCI, vol. 252, pp. 31–63. Springer, Heidelberg (2009)
6. Ławryńczuk, M.: Explicit nonlinear predictive control of a distillation column based on neural models. Chemical Engineering and Technology 32, 1578–1587 (2009)
7. Ławryńczuk, M.: A family of model predictive control algorithms with artificial neural networks. International Journal of Applied Mathematics and Computer Science 17, 217–232 (2007)
8. Maciejowski, J.M.: Predictive control with constraints. Prentice Hall, Harlow (2002)
9. Maner, B.R., Doyle, F.J., Ogunnaike, B.A., Pearson, R.K.: Nonlinear model predictive control of a simulated multivariable polymerization reactor using second-order Volterra models. Automatica 32, 1285–1301 (1996)
10. Morari, M., Lee, J.H.: Model predictive control: past, present and future. Computers and Chemical Engineering 23, 667–682 (1999)
11. Nørgaard, M., Ravn, O., Poulsen, N.K., Hansen, L.K.: Neural networks for modelling and control of dynamic systems. Springer, London (2000)
12. Pillay, N., Govender, P.: Particle swarm optimization of PID tuning paremeters: optimal tuning of single-input-single-output control loops. LAP Lambert Academic Publishing (2010)
13. Pourjafari, E., Mojallali: Predictive control for voltage collapse avoidance using a modified discrete multi-valued PSO algorithm. ISA Transactions 50, 195–200 (2011)
14. Qin, S.J., Badgwell, T.A.: A survey of industrial model predictive control technology. Control Engineering Practice 11, 733–764 (2003)
15. Tatjewski, P.: Advanced control of industrial processes, Structures and algorithms. Springer, London (2007)
16. Yousuf, M.S.: Nonlinear predictive control using particle swarm optimization: application to power systems. VDM Verlag (2010)