Acceleration of Nonlinear POD Models: A Neural Network Approach

Oscar Mauricio Agudelo, Jairo José Espinosa, and Bart De Moor

Abstract— **This paper presents a way of accelerating the evaluation and simulation of nonlinear POD models by using feedforward neural networks. Traditionally, Proper Orthogonal Decomposition (POD) and Galerkin projection have been employed to reduce the high-dimensionality of the discretized systems used to approximate Partial Differential Equations (PDEs). Although a large model-order reduction can be obtained with these techniques, the computational saving is small when we are dealing with nonlinear or Linear Time Variant (LTV) models. If we approximate the nonlinear vector function of the POD models by means of a feedforward neural network like a Multi-Layer Perceptron (MLP), then we can speed up the simulation of the POD models given that the on-line evaluation of this kind of networks can be done very fast. This is the approach that is presented in this paper.**

I. INTRODUCTION

Proper Orthogonal Decomposition (POD), also known as Karhunen-Loève expansion or Principal Component Analysis (PCA), is a technique that has been applied in many physical systems modeled by Partial Differential Equations (PDEs) for deriving reduced order models. POD is a data driven method where a suitable set of orthonormal basis vectors are derived from simulation or experimental data. These basis vectors, which are organized in order of relevance, capture the spatial dynamics of the system. The reduced order model is obtained by projecting (Galerkin projection) the high-dimensional model on the most relevant basis vectors.

Although we can reach a large model-order reduction with the POD technique, this reduction does not lead to a significant computational saving when nonlinear or Linear Time Variant (LTV) models are considered. The reason of this limitation lies in the fact that we need the full spatial information from the original high-dimensional systems in order to evaluate the reduced order models. In [1][2][3] a method known as Missing Point Estimation (MPE) is proposed for tackling this problem. In this method the Galerkin projection is conducted only on some pre-selected state variables instead of the entire set. The remaining state variables are estimated by means of the POD basis vectors. It has been reported that this technique can save considerable computational effort.

In this paper we propose an alternative way of accelerating nonlinear POD models. In our approach, we approximate the nonlinear vector function of the POD models by means of a Multi-Layer Perceptron (MLP). In this way, we eliminate the necessity of having the full spatial information from the original models. This technique can lead to a significant saving of computational time given that the MLP can be evaluated very fast.

The paper is organized as follows. Section II presents a description of the dynamical system that will be used for explaining our technique, the heat transfer along a one-dimensional bar. In section III the derivation of a reduced order model for the bar using POD and Galerkin projection is discussed. Section IV introduces our approach for accelerating the POD model found in section III. In Section V we show some validation and simulation results and finally in Section VI we summarize the main conclusions.

O. M. Agudelo is with the Department of Electrical Engineering (ESAT), Research Group SCD-SISTA, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee (Leuven), Belgium. He is also with the Department of Automation and Electronics, Universidad Autónoma de Occidente, Calle 25 No 115-85, Cali, Colombia. mauricio.agudelo@esat.kuleuven.be

J. J. Espinosa is with the Facultad de Minas, Universidad Nacional de Colombia, Carrera 80 No. 65-223, Medellin, Colombia. jairo.espinosa@ieee.org

B. De Moor is with the Department of Electrical Engineering (ESAT), Research Group SCD, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, B-3001 Heverlee (Leuven), Belgium. bart.demoor@esat.kuleuven.be

II. HEAT TRANSFER IN A ONE-DIMENSIONAL BAR

The system under study is the silicon bar shown in Figure 1. The bar has attached an actuator which provides a uniformly distributed heat flux $u(t)$ between $x = x_{\mathrm{a}}$ and $x = x_{\mathrm{b}}$. Additionally, an external heat flux $d(t)$ is applied uniformly along the bar whose ends are kept at $25^{\circ}\mathrm{C}$ (ambient temperature) all the time.

If only temperature variations in the x-direction are considered, the dynamics of the temperature $T(x,t)$ of the bar can be modeled by the following nonlinear PDE:

$$\rho C_{\mathrm{p}} \frac{\partial T(x,t)}{\partial t} = \frac{\partial}{\partial x}\left(\kappa\left(T(x,t)\right)\frac{\partial T(x,t)}{\partial x}\right) + V(x,t) \quad (1)$$

where ρ is the material density in $[\mathrm{Kg}\cdot\mathrm{m}^{-3}]$, C_{p} is the heat capacity in $[\mathrm{J}\cdot\mathrm{Kg}^{-1}\cdot\mathrm{K}^{-1}]$, $\kappa(T)$ is the temperature dependent heat conductivity in $[\mathrm{J}\cdot\mathrm{s}^{-1}\cdot\mathrm{m}^{-1}\cdot\mathrm{K}^{-1}]$, t is the time in $[\mathrm{s}]$, x is the spatial coordinate in $[\mathrm{m}]$ and $V(x,t)$ is the heat source applied to the bar at position x and time t in $[\mathrm{W}\cdot\mathrm{m}^{-3}]$. $V(x,t)$ is defined as follows:

$$V(x,t) = \begin{cases} d(t)+u(t), & x_{\mathrm{a}} \le x \le x_{\mathrm{b}} \\ d(t), & \text{elsewhere} \end{cases}.$$

The relation between the temperature and the heat conductivity $\kappa(T)$ is described by a polynomial of degree 3,

$$\kappa(T) = \kappa_0 + \kappa_1 T + \kappa_2 T^2 + \kappa_3 T^3 \quad (2)$$

where $\kappa_0 = 36$, $\kappa_1 = -0.1116$, $\kappa_2 = 1.7298\times10^{-4}$ and $\kappa_3 = -1.78746\times10^{-7}$ are real coefficients in the appropriated units.

ISBN 978-963-311-369-1
© Copyright EUCA 2009

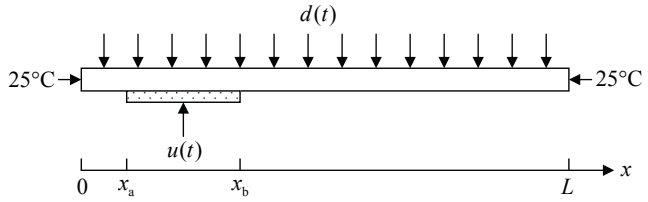

Fig. 1. Silicon Bar.

The initial and boundary conditions (dirichlet) of (1) are given by: $T(x,0) = 25°\mathrm{C}$, $T(x=0,t) = T(x=L,t) = 25°\mathrm{C}$. The length of the bar is $L = 0.1$ m and the remaining numerical values of the model parameters are: $\rho = 3970\ \mathrm{Kg}\cdot\mathrm{m}^{-3}$, $C_\mathrm{p} = 766\ \mathrm{J}\cdot\mathrm{Kg}^{-1}\cdot\mathrm{K}^{-1}$, $x_\mathrm{a} = 0.005$ m and $x_\mathrm{b} = 0.04$ m. The operating ranges in $[\mathrm{W}\cdot\mathrm{m}^{-3}]$ of $d(t)$ and $u(t)$ are $-500\cdot 10^3 \leq d(t) \leq 500\cdot 10^3$ and $-1500\cdot 10^3 \leq u(t) \leq 1500\cdot 10^3$ respectively. Some of the previous numerical values were inspired on the values given in [4] .

For simulation purposes, it is necessary to reduce the infinity dimensionality of (1) by discretizing the spatial domain. To this end, the partial derivatives with respect to space were replaced by backward (the inner spatial derivative) and forward (the outer spatial derivative) difference approximations. This is equivalent to replace the second partial derivative with respect to space by a central difference approximation in the linear version of the heat equation where κ is kept constant. The discretized model of the bar is given by the following set of nonlinear Ordinary Differential Equations (ODEs):

$$\frac{dT_i}{dt} = c_1\Big(\kappa(T_{i+1})T_{i+1} - \left(\kappa(T_{i+1}) + \kappa(T_i)\right)T_i + \\ +\kappa(T_i)T_{i-1}\Big) + c_2 V_i \tag{3}$$

$$c_1 = \frac{1}{\rho C_\mathrm{p}\left(\Delta x\right)^2}, \quad c_2 = \frac{1}{\rho C_\mathrm{p}}, \quad T_0 = T_N = 25°\mathrm{C}$$

$$\text{for } i = 1,\ldots,N-1,$$

where N is the number of sections in which the bar is divided, Δx is the length of each section, and T_i and V_i are the temperature and heat flux at the point $x_i = i\Delta x$.

If we define $\mathbf{T}(t) \in \mathbb{R}^{N-1}$ as the vector containing the temperature of the grid points from x_1 till x_{N-1} every time instant, we can write equation (3) as follows:

$$\dot{\mathbf{T}}(t) = \mathbf{F}\left(\mathbf{T}(t)\right) + \mathbf{B}_1 d(t) + \mathbf{B}_2 u(t) \tag{4}$$

where $\mathbf{F}\left(\mathbf{T}(t)\right) : \mathbb{R}^{N-1} \rightarrow \mathbb{R}^{N-1}$ is a vector-valued or vector function which contains the nonlinear terms of the model, and $\mathbf{B}_1$ and $\mathbf{B}_2$ are vectors defined as: $\mathbf{B}_1 \in \mathbb{R}^{N-1} = (c_2, c_2, \ldots, c_2)^T$, $\mathbf{B}_2 \in \mathbb{R}^{N-1} = (0,\ldots,0,c_2,\ldots,c_2,0,\ldots,0)^T$. The position of the nonzero elements in $\mathbf{B}_2$ corresponds to the position of the grid points that are in contact with the actuator.

The spatial domain was divided into $N = 500$ sections which means that (4) has $N-1 = 499$ states. With such amount of states the design of a control system for the bar is not an easy task. In addition, the simulation of (4) is a task that demands a considerable amount of computational resources. A way of improving this situation is by finding a reduced order model (few number of equations and states) that approximates (4) by means of a technique known as proper orthogonal decomposition which is discussed in the next section.

III. Model Reduction using POD

Given that the initial state of (4) does not provide information about the system dynamics, we are going to work with the temperature deviations with respect to the ambient temperature ($25°\mathrm{C}$). Consequently, the vector $\mathbf{T}(t)$ is split as follows:

$$\mathbf{T}(t) = \mathbf{T}^\Delta(t) + \mathbf{T}^*$$

where $\mathbf{T}^\Delta(t) \in \mathbb{R}^{N-1}$ is the vector containing the deviations of the temperature profile and $\mathbf{T}^* \in \mathbb{R}^{N-1}$ is a constant vector which contains the initial temperature profile of the bar (ambient temperature).

In POD, we start by observing that $\mathbf{T}^\Delta(t) \in \mathbb{R}^{N-1}$ can be expanded as a sum of orthonormal basis vectors:

$$\mathbf{T}^\Delta(t) = \sum_{j=1}^{N-1} a_j(t)\boldsymbol{\varphi}_j \tag{5}$$

where $\boldsymbol{\varphi}_j \in \mathbb{R}^{N-1}\ \forall j = 1,\ldots,N-1$ is a set of orthonormal basis vectors (POD basis vectors or POD basis functions) in the discretized spatial domain, and $a_j(t) \in \mathbb{R}\ \forall j = 1,\ldots,N-1$ are the time-varying coefficients, or POD coefficients, associated to each basis vector. These POD basis vectors are ordered according to their relevance to $\mathbf{T}^\Delta(t)$.

The main dynamics of the system can be represented using the first n most relevant basis vectors, since $\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2, \ldots, \boldsymbol{\varphi}_n$ condensates the main spatial correlations. An nth order approximation of (5) is then given by the truncated sequence

$$\mathbf{T}^\Delta_n(t) = \sum_{j=1}^{n} a_j(t)\boldsymbol{\varphi}_j, \quad n \ll N-1. \tag{6}$$

An approximate (reduced order) model of $\mathbf{T}^\Delta(t)$ can be derived by building a model for the first n POD coefficients. This is the essence of model reduction by POD.

The POD basis vectors are determined from simulation or experimental data of the process. The dynamic model for the first n POD coefficients is commonly found by means of the Galerkin projection [1]. For the specific case of the linear systems, subspace identification techniques can be used for finding such a model [5][6].

In the next subsections we present in detail the steps followed for deriving the reduced order model of the bar.

A. Generation of the snapshot Matrix

We have built the snapshot matrix $\mathbf{T}_\mathrm{snap} \in \mathbb{R}^{499\times 2001}$ by collecting the evolution of the deviations of the temperature profile when Pseudo Random Multilevel Noise Signals (PRMNS) were applied to the process inputs $u(t)$ and $d(t)$,

$$\mathbf{T}_\mathrm{snap} = (\mathbf{T}^\Delta(0), \mathbf{T}^\Delta(\Delta t), \ldots, \mathbf{T}^\Delta(2000\Delta t)).$$

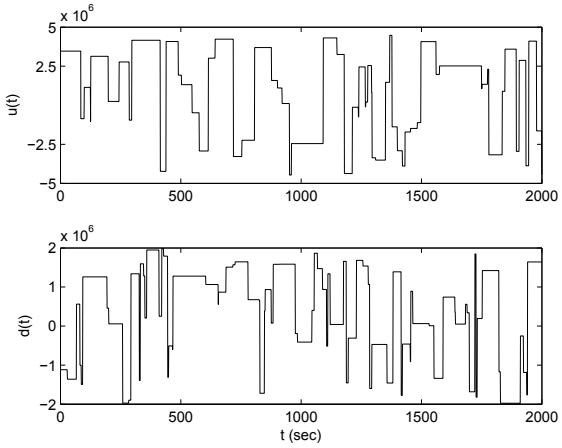

Fig. 2. Pseudo Random Multilevel Noise Signals (PRMNS) used in the generation of the snapshot matrix $\mathbf{T}_{\text{snap}}$. Amplitudes in $[\text{W}\cdot\text{m}^{-3}]$.

These excitation signals can be observed in Figure 2. A commutation probability of 3% was set for the signals and the amplitudes in $[\text{W}\cdot\text{m}^{-3}]$ of $d(t)$ and $u(t)$ were restricted to the intervals $[-500\cdot10^3\ \ 500\cdot10^3]$ and $[-1500\cdot10^3\ \ 1500\cdot10^3]$ respectively. Along the simulations, 2001 samples were collected using a sampling time Δt of 1 s.

B. Derivation of the POD basis Vectors

The POD basis vectors were derived by calculating the singular value decomposition (SVD) of $\mathbf{T}_{\text{snap}}$,

$$\mathbf{T}_{\text{snap}} = \mathbf{\Phi}\mathbf{\Sigma}\mathbf{\Psi}^T$$

where $\mathbf{\Phi} \in \mathbb{R}^{499\times499}$ and $\mathbf{\Psi} \in \mathbb{R}^{2001\times2001}$ are unitary matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{499\times2001}$ is a matrix that contains the singular values of $\mathbf{T}_{\text{snap}}$ in a decreasing order on its main diagonal. The left singular vectors, that is the columns of $\mathbf{\Phi}$

$$\mathbf{\Phi} \in \mathbb{R}^{499\times499} = (\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2, \ldots, \boldsymbol{\varphi}_{499})$$

are the POD basis vectors.

C. Selection of the most relevant POD basis vectors

We have made the selection by checking the singular values of $\mathbf{T}_{\text{snap}}$, the larger the singular value the more relevant the basis vector is. In this problem, the first 6 basis vectors were selected. The 6th order approximation of $\mathbf{T}^{\Delta}(t)$ is given by

$$\mathbf{T}_6^{\Delta}(t) = \sum_{j=1}^{6} a_j(t)\boldsymbol{\varphi}_j = \mathbf{\Phi}_6\mathbf{a}(t), \tag{7}$$

where $\mathbf{\Phi}_6 = (\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_6)$ and $\mathbf{a}(t) = (a_1(t), \ldots, a_6(t))^T$.

D. Construction of the model for the POD coefficients

In order to derive a dynamic model for the POD coefficients, we have used the Galerkin projection. If we define a residual function $R(\mathbf{T}, \dot{\mathbf{T}})$ for equation (4) as follows:

$$R(\mathbf{T}, \dot{\mathbf{T}}) = \dot{\mathbf{T}}(t) - \mathbf{F}(\mathbf{T}(t)) - \mathbf{B}_1 d(t) - \mathbf{B}_2 u(t) \tag{8}$$

and we replace $\mathbf{T}(t)$ by its 6th order approximation $\mathbf{T}_6(t) = \mathbf{T}_6^{\Delta} + \mathbf{T}^*$ in (8), the projection of $R(\mathbf{T}_6, \dot{\mathbf{T}}_6)$ on the space spanned by the basis vectors $\mathbf{\Phi}_6$ shall vanish. That is,

$$\left\langle R(\mathbf{T}_6, \dot{\mathbf{T}}_6), \boldsymbol{\varphi}_j \right\rangle = 0; \quad j = 1, \ldots, n = 6 \tag{9}$$

where $\langle .,. \rangle$ denotes the Euclidean inner product. If we replace $\mathbf{T}(t)$ by its 6th order approximation $\mathbf{T}_6(t) = \mathbf{\Phi}_6\mathbf{a}(t) + \mathbf{T}^*$ in equation (4), and we apply the inner product criterion (9) to the resulting equation,

$$\mathbf{\Phi}_6^T\mathbf{\Phi}_6\dot{\mathbf{a}}(t) = \mathbf{\Phi}_6^T\mathbf{F}\left(\mathbf{\Phi}_6\mathbf{a}(t) + \mathbf{T}^*\right) + \mathbf{\Phi}_6^T\mathbf{B}_1 d(t) + \mathbf{\Phi}_6^T\mathbf{B}_2 u(t)$$

$$\dot{\mathbf{a}}(t) = \mathbf{\Phi}_6^T\mathbf{F}\left(\mathbf{\Phi}_6\mathbf{a}(t) + \mathbf{T}^*\right) + \mathbf{\Phi}_6^T\mathbf{B}_1 d(t) + \mathbf{\Phi}_6^T\mathbf{B}_2 u(t)$$

then we can find the reduced order model of the bar with only 6 states,

$$\dot{\mathbf{a}}(t) = \mathbf{\Phi}_6^T\mathbf{F}\left(\mathbf{\Phi}_6\mathbf{a}(t) + \mathbf{T}^*\right) + \tilde{\mathbf{B}}_1 d(t) + \tilde{\mathbf{B}}_2 u(t) \tag{10}$$

where $\tilde{\mathbf{B}}_1 = \mathbf{\Phi}_6^T\mathbf{B}_1$ and $\tilde{\mathbf{B}}_2 = \mathbf{\Phi}_6^T\mathbf{B}_2$. Finally if we define a new vector valued function $\mathbf{f} : \mathbb{R}^6 \to \mathbb{R}^6$ as $\mathbf{f}\left(\mathbf{a}(t)\right) = \mathbf{\Phi}_6^T\mathbf{F}\left(\mathbf{\Phi}_6\mathbf{a}(t) + \mathbf{T}^*\right)$, then the reduced order model of the bar can be written more compactly as follows:

$$\dot{\mathbf{a}}(t) = \mathbf{f}\left(\mathbf{a}(t)\right) + \tilde{\mathbf{B}}_1 d(t) + \tilde{\mathbf{B}}_2 u(t). \tag{11}$$

IV. Acceleration of POD models by using Neural Networks

The POD and Galerkin projection techniques have been successfully used in many applications where it is required a significant model-order reduction. Unlike the Linear Time Invariant (LTI) case, this model-order reduction does not lead to an important saving of computational effort for the nonlinear and Linear Time Variant (LTV) cases.

In this section we present a way of speeding up the evaluation of POD models from nonlinear systems like (4) by using feedforward neural networks.

In general, it should be clear that we do not know the compact expression of $\mathbf{f}(\mathbf{a}(t))$ in (11). So, in order to simulate the reduced order model, the ODE solver has to evaluate indirectly $\mathbf{f}(\mathbf{a}(t))$. Firstly, the solver has to map the state of the reduced order model $\mathbf{a}(t)$ into the original high-dimensional space by means of this linear transformation $\mathbf{T}_6^{\Delta}(t) = \mathbf{\Phi}_6\mathbf{a}(t)$. Secondly it has to evaluate the resulting high-dimensional state vector $\mathbf{T}_6^{\Delta}(t)$ in the vector function $\mathbf{F}(\mathbf{T}_6^{\Delta}(t) + \mathbf{T}^*)$ of (4), and finally it has to map the results of this evaluation to the low dimensional space by pre-multiplying them by $\mathbf{\Phi}_6^T$. The evaluation of $\mathbf{f}(\mathbf{a}(t))$ is done as many times as it is required by the ODE solver within each integration step. Hence the indirect evaluation of $\mathbf{f}(\mathbf{a}(t))$ is the bottleneck that limits severely the computational gain of the POD model.

In order to tackle this problem, we propose to approximate the vector function $\mathbf{f} : \mathbb{R}^6 \to \mathbb{R}^6$ by using a Multi-Layer Perceptron (MLP) neural network. In this way we eliminate the necessity of evaluating the vector function $\mathbf{F} : \mathbb{R}^{499} \to \mathbb{R}^{499}$ of the full order model and we can save a considerable amount of time. As it is well-known, a multi-layer perceptron can learn any nonlinear input-output mapping

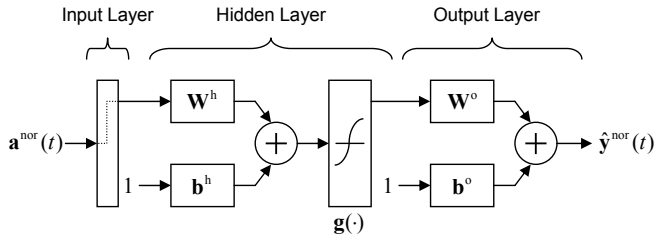

Fig. 3. Structure of the Multi-layer Perceptron. $\mathbf{a}^{\text{nor}}(t) \in \mathbb{R}^6$ and $\hat{\mathbf{y}}^{\text{nor}}(t) \in \mathbb{R}^6$

given an adequate number of hidden neurons (each one with a nonlinear activation function) in its hidden layers [7]. In addition, the time required for calculating the MLP output can be quite short since only few matrix multiplications, vector additions and function evaluations are necessary. Due to these characteristics, an MLP is a suitable choice for approximating the vector function **f** in (11).

In order to generate the input and output data required for training, validating and testing the MLP, firstly the POD model (11) was excited with PRMNS signals and the evolution in time of the state vector **a**(t) was collected. From the test the following data sets were constructed:

$$\mathcal{A} = \{\mathbf{a}(0), \mathbf{a}(\Delta t), \ldots, \mathbf{a}(10000\Delta t)\},$$
$$\mathcal{U} = \{u(0), u(\Delta t), \ldots, u(10000\Delta t)\},$$
$$\mathcal{D} = \{d(0), d(\Delta t), \ldots, d(10000\Delta t)\}.$$

In the experiment 10001 samples were gathered with a sampling time Δt equal to 1 s. The commutation probability of the PRMNS signals was set to 3% and the amplitudes in $[\mathrm{W} \cdot \mathrm{m}^{-3}]$ of $d(t)$ and $u(t)$ were restricted to the intervals $[-600 \cdot 10^3 \quad 600 \cdot 10^3]$ and $[-1800 \cdot 10^3 \quad 1800 \cdot 10^3]$ respectively. Notice that these intervals are 20% larger than the operating ranges defined in section II. This enlarges the range of validity of our approximation with the MLP.

If we define a vector $\mathbf{y}(t) \in \mathbb{R}^6$ as follows:

$$\mathbf{y}(t) = \dot{\mathbf{a}}(t) - \tilde{\mathbf{B}}_1 d(t) - \tilde{\mathbf{B}}_2 u(t), \tag{12}$$

then (11) can be cast as $\mathbf{y}(t) = \mathbf{f}(\mathbf{a}(t))$. By using (11), and the data sets $\mathcal{U}$, $\mathcal{D}$ and $\mathcal{A}$, we can easily calculate $\dot{\mathbf{a}}(t)$ at each sampling time and afterwards $\mathbf{y}(t)$ by means of (12). The evolution in time of $\mathbf{y}(t)$ is then compiled in the following data set,

$$\mathcal{Y} = \{\mathbf{y}(0), \mathbf{y}(\Delta t), \ldots, \mathbf{y}(10000\Delta t)\}.$$

In order to make the training of the MLP more efficient, the input data $\mathcal{A}$ and the target outputs $\mathcal{Y}$ were normalized for zero mean and unit variance by applying the normalization functions $\mathbf{h} : \mathbb{R}^6 \to \mathbb{R}^6$ and $\mathbf{v} : \mathbb{R}^6 \to \mathbb{R}^6$ to each element of the data sets $\mathcal{A}$ and $\mathcal{Y}$ respectively. The ith component function of the vector functions $\mathbf{h}$ and $\mathbf{v}$ is defined as

$$a_i^{\text{nor}}(t) = h_i(\mathbf{a}(t)) = \frac{a_i(t) - \bar{a}_i}{\sigma_{a_i}}, \tag{13}$$

$$y_i^{\text{nor}}(t) = v_i(\mathbf{y}(t)) = \frac{y_i(t) - \bar{y}_i}{\sigma_{y_i}}, \tag{14}$$

where $\bar{a}_i$, $\bar{y}_i$ and σ_{a_i}, σ_{y_i} are the mean and the standard deviation of $a_i(t)$ and $y_i(t)$ respectively. At the moment of using the MLP after training, the input data has to be normalized by using (13) and the output of the neural network needs to be restored using the inverse function of (14) whose ith component function is defined as follows:

$$\hat{y}_i(t) = v_i^{-1}(\hat{\mathbf{y}}^{\text{nor}}(t)) = \hat{y}_i^{\text{nor}}(t)\sigma_{y_i} + \bar{y}_i.$$

Here the "hat" on top of y_i and y_i^{nor} are used to stress that the output of the MLP is just an approximation of the target output. The matrices containing the normalized input data and output targets are denoted as $\mathcal{A}^{\text{nor}}$ and $\mathcal{Y}^{\text{nor}}$ respectively.

The structure of the MLP neural network is presented in Figure 3. The MLP has only 1 hidden layer with $N_{\text{hn}} = 10$ hidden neurons. The input layer and the output layer have 6 neurons each. The output of the MLP is given by the following expression:

$$\hat{\mathbf{y}}^{\text{nor}}(t) = \mathbf{W}^{\text{o}} \cdot \mathbf{g}\left(\mathbf{W}^{\text{h}} \cdot \mathbf{a}^{\text{nor}}(t) + \mathbf{b}^{\text{h}}\right) + \mathbf{b}^{\text{o}} \tag{15}$$

where $\mathbf{W}^{\text{h}} \in \mathbb{R}^{10\times 6}$ is the matrix of weights that links the input layer to the hidden layer, the entry W_{ji}^{h} of $\mathbf{W}^{\text{h}}$ corresponds to the connection weight from the ith input neuron to the jth neuron in the hidden layer, $\mathbf{b}^{\text{h}} \in \mathbb{R}^{10}$ is the vector containing the bias weight of each neuron of the hidden layer, $\mathbf{W}^{\text{o}} \in \mathbb{R}^{6\times 10}$ is the matrix of weights that links the hidden layer to the output layer, the entry W_{ji}^{o} of $\mathbf{W}^{\text{o}}$ is the connection weight from the ith hidden neuron to the jth neuron in the output layer, $\mathbf{b}^{\text{o}} \in \mathbb{R}^6$ is the vector that contains the bias weight of each neuron of the output layer, and $\mathbf{g}(\cdot) : \mathbb{R}^{10} \to \mathbb{R}^{10}$ is a vector-valued function whose component functions are the nonlinear activation functions of the hidden neurons. The ith component function of $\mathbf{g}(\cdot)$ is a hyperbolic tangent function which is defined by the following equation

$$g_i(\mathbf{s}) = \frac{e^{2s_i} - 1}{e^{2s_i} + 1},$$

where $\mathbf{s} \in \mathbb{R}^{10}$ is the vector containing the weighted sum of each hidden neuron. The MLP was trained by using the Levenberg-Marquardt (LM) backpropagation algorithm which is available in the Matlab Neural Network Toolbox. In general, this algorithm offers a good velocity of convergence and acceptable memory requirements when it is used for approximating functions with networks that contain up to a few hundred weights. In order to avoid the overfitting of the MLP, the early stopping method was used during the training, and therefore the data ($\mathcal{A}^{\text{nor}}$ and $\mathcal{Y}^{\text{nor}}$) was divided into 3 sets: the training set with 7001 data points, the validation set with 1500 data points and the test set with 1500 data points. The training set is used by the training algorithm for updating the network weights and biases, the validation set is used for detecting the overfitting during the training stage and the test set is used for testing the generalization capabilities of the MLP. The test set is never used during the training stage.

The data was divided by cycling samples (interleaved data division) between the training set, validation set, and test set according to percentages. These percentages were 70%, 15%

TABLE I
PERFORMANCE OF THE POD MODELS

Test	POD model			Neural-POD model		
	G_{d}	G_{s}	$\Delta T_{\max}[^{\circ}\mathrm{C}]$	G_{d}	G_{s}	$\Delta T_{\max}[^{\circ}\mathrm{C}]$
1	0.97	2.02	0.423	9.66	8.153	0.689
2	0.97	3.61	0.279	9.32	11.69	0.278
3	0.97	4.52	0.0518	9.32	14.03	0.0519
4	0.97	3.52	0.0594	9.30	11.43	0.0593

G_d is the computational gain in the calculation of the derivatives.
G_s is the computational gain in the simulation of the model.
$\Delta T_{\max}$ is the largest temperature deviation (error) of the POD model.

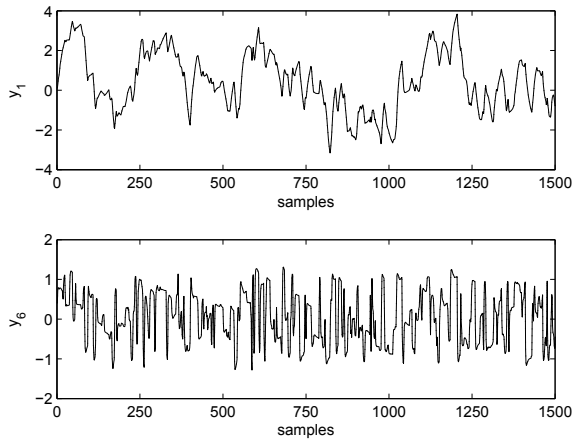

Fig. 4. MLP test performance for the outputs $y_1(t)$ and $y_6(t)$. Solid line - Data points (Targets). Dashed line - MLP.

and 15% for the training, validation and test sets respectively. The Mean Squared Error (MSE) function was selected to measure the performance of the neural network. After the training stage, the MSE of the MLP for the training set was 8.0662×10^{-7} and for the test set was 8.0975×10^{-7}. The training of the MLP required about 6000 epochs to achieve these MSE errors. Figure 4 presents the MLP output and the original data points (targets) for $y_1(t)$ and $y_6(t)$ when the test set is used. The MLP output is practically overlapping the data points, and it is really difficult to see any difference. It is clear that the network has learned the nonlinear input-output mapping $\mathbf{f}$ with a high degree of accuracy, and additionally the net has shown a good generalization capability. One factor that contributes to have very small MSE errors is the absence of noise in the data.

Finally, the equation of the POD model where the function $\mathbf{f}$ has been approximated by an MLP is the following one:

$$\dot{\mathbf{a}}(t) = \hat{\mathbf{f}}\left(\mathbf{a}(t)\right) + \tilde{\mathbf{B}}_1 d(t) + \tilde{\mathbf{B}}_2 u(t) \tag{16}$$

with $\hat{\mathbf{f}}\left(\mathbf{a}(t)\right) = \mathbf{v}^{-1}\left(\mathbf{W}^{\mathrm{o}} \cdot \mathbf{g}\left(\mathbf{W}^{\mathrm{h}} \cdot \mathbf{h}\left(\mathbf{a}(t)\right) + \mathbf{b}^{\mathrm{h}}\right) + \mathbf{b}^{\mathrm{o}}\right)$. From now on, this POD model will be referred to as Neural-POD model.

V. SIMULATIONS RESULTS

In order to validate and evaluate the POD models found in sections III and IV, the following tests were carried out:

- Test 1: A step of magnitude $1200{\cdot}10^3\ \mathrm{W}{\cdot}\mathrm{m}^{-3}$ is applied to $u(t)$ and a step of magnitude $500 \cdot 10^3\ \mathrm{W} \cdot \mathrm{m}^{-3}$ is applied to $d(t)$.
- Test 2: Steps of magnitude $-1100 \cdot 10^3\ \mathrm{W} \cdot \mathrm{m}^{-3}$ and $-400 \cdot 10^3\ \mathrm{W} \cdot \mathrm{m}^{-3}$ are applied to $u(t)$ and $d(t)$ respectively.
- Test 3: A step of magnitude $500{\cdot}10^3\ \mathrm{W}{\cdot}\mathrm{m}^{-3}$ is applied to $u(t)$ and a step of magnitude $-200 \cdot 10^3\ \mathrm{W} \cdot \mathrm{m}^{-3}$ is applied to $d(t)$.
- Test 4: Steps of magnitude $-400{\cdot}10^3\ \mathrm{W}{\cdot}\mathrm{m}^{-3}$ and $300{\cdot}10^3\ \mathrm{W} \cdot \mathrm{m}^{-3}$ are applied to $u(t)$ and $d(t)$ respectively.

Given that (4) is a stiff equation, an ODE solver that can deal with this has to be used. Hence, we used the function *ode23tb* of Matlab which implements TR-BDF2, an implicit Runge-Kutta formula with a first stage that is a trapezoidal rule step and a second stage that is a backward differentiation formula of order two [8]. Along this work, not only (4) was solved with *ode23tb*, but also (11) and (16). The solver was configured with a variable integration step and with a relative tolerance of 10^{-5} in all the cases. The initial conditions for the POD models were given by $\mathbf{a}(0) = \mathbf{\Phi}_6^T \mathbf{T}_6^{\Delta}(0) = \mathbf{0}$.

In Table I we present the computational gain of the POD models with respect to (4) and a measure of their accuracy. In this table $\Delta T_{\max}$ is the largest temperature deviation (error) of the POD models with respect to the full order model along the entire test, and G_{d} and G_{s} quantify the computational gain of the POD models with respect to the full order model. They are defined as follows:

$$G_{\mathrm{d}} = {\tilde{t}_{\mathrm{fom}}}/{\tilde{t}_{\mathrm{pod}}}, \quad G_{\mathrm{s}} = {t_{\mathrm{fom}}}/{t_{\mathrm{pod}}}$$

where t_{fom} and t_{pod} are the times spent by the ODE solver for simulating the full order model and the POD model respectively, and $\tilde{t}_{\mathrm{fom}}$ and $\tilde{t}_{\mathrm{pod}}$ are the average times for calculating the derivatives along the test of the full order model and the POD model respectively. The values of G_{d} and G_{s} in Table I are average values found after 15000 and 1500 runs respectively.

Figures 5 and 6 show the maximum temperature deviation of the POD models with respect to the full order model along the tests, and Figure 7 depicts the evolution of the temperature profile of the bar during test 1 (the most severe test). From these figures it is clear that the accuracy of the POD models is good in spite of the big model-order reduction. Also, the difference between the POD model and the Neural-POD model is very small. The largest difference between them occurs during test 1 and this difference is merely of $0.26^{\circ}\mathrm{C}$. In Figure 7 we can observe how the temperature profiles of both POD models overlap each other.

From Table I we can notice that the derivatives of the Neural-POD model are calculated about 9.7 times faster than in the POD model. This is a significant gain that have been achieved thanks to the use of an MLP. This gain has a positive impact on the simulation time. Hence the simulation of the Neural-POD model requires about 3.4 times less time than the simulation of the POD model. To sum up, the approximation of the vector function $\mathbf{f}$ in (11) by an MLP

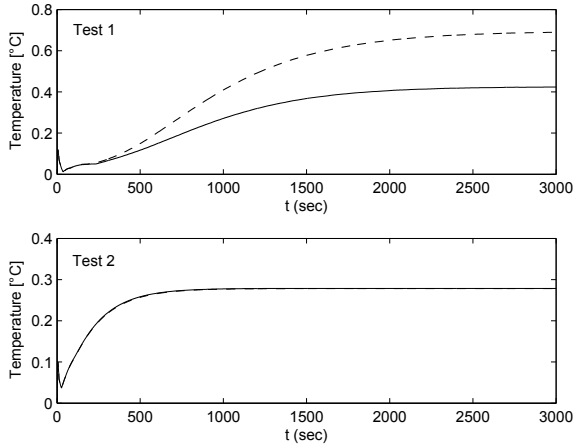

Fig. 5. Maximum temperature deviation of the POD models with respect to the full order model along Tests 1 and 2. Solid line - POD model. Dashed line - Neural-POD model.

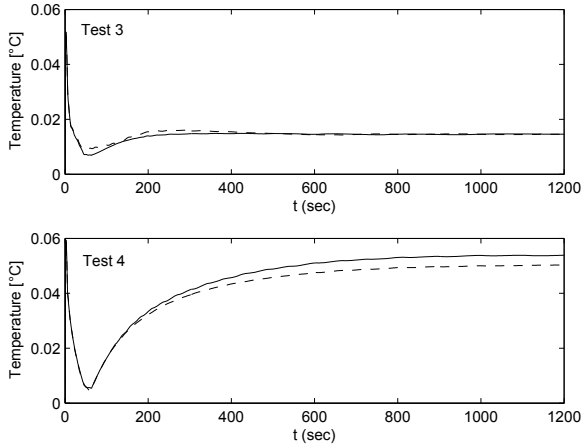

Fig. 6. Maximum temperature deviation of the POD models with respect to the full order model along Tests 3 and 4. Solid line - POD model. Dashed line - Neural-POD model.

has led to speeding up the evaluation of this function and therefore to reduce the simulation time of the POD model.

VI. Conclusions

In this paper, we have presented a way of speeding up the evaluation of a nonlinear POD model by using a multi-layer perceptron. In this approach, the nonlinear vector function of the POD model is approximated by an MLP which in general can be evaluated much faster than the original vector function. The computational gain that we can obtain is then limited by the size of the MLP. The larger the MLP the lesser is the computational gain. However, for large MLPs we can expect a better accuracy than for the small ones, and a better capability of learning complex nonlinear mappings. So, the size of the net has to be chosen in such a way that provides a good trade-off between accuracy and computational gain.

The accuracy of the neural-POD model of the bar was as good as the accuracy of the original POD model, but it demanded less computational effort. Further research is

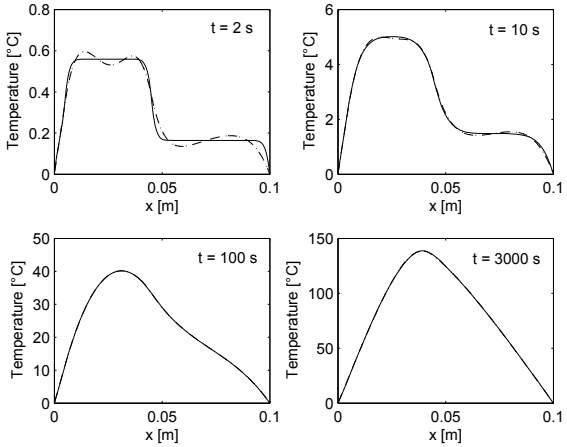

Fig. 7. Temperature profile at different time steps. Solid line - Full order model (4). Dashed line - POD model. Dotted line - Neural-POD model. The temperatures are relative to the ambient temperature (25°C).

necessary in order to evaluate the approach proposed in this paper on dynamic systems with harder nonlinearities.

VII. Acknowledgments

This research was supported by: ● Research Council KUL: GOA AMBioRICS, CoE EF/05/006 Optimization in Engineering (OPTEC), IOF-SCORES4CHEM, several PhD/postdoc & fellow grants; ● Flemish Government: ○ FWO: PhD/postdoc grants, projects G.0452.04 (new quantum algorithms), projects G.0499.04 (Statistics), G.0211.05 (Nonlinear), G.0226.06 (cooperative systems and optimization), G.0321.06 (Tensors), G.0302.07 (SVM/Kernel), G.0320.08 (convex MPC), G.0558.08 (Robust MHE), G.0557.08 (Glycemia2), research communities (ICCoS, ANMMM, MLDM); ○IWT: PhD Grants, McKnow-E, Eureka-Flite+; ○Helmholtz: viCERP; ● Belgian Federal Science Policy Office: IUAP P6/04 (DYSCO, Dynamical systems, control and optimization, 2007-2011); ● EU: ERNSI; FP7-HD-MPC (Collaborative Project STREP-grantnr. 223854) ● Contract Research: AMINAL. Dr. Bart De Moor is a full professor at the Katholieke Universiteit Leuven, Belgium.

References

[1] P. Astrid, "Reduction of process simulation models: a proper orthogonal decomposition approach," Ph.D. dissertation, Technishche Universiteit Eindhoven, Eindhoven (Netherlands), November 2004.

[2] P. Astrid, S. Weiland, K. Willcox, and T. Backx, "Missing point estimation in models described by proper orthogonal decomposition," in *Proceedings of the 43rd IEEE Conference on Decision and Control*, Bahamas, December 2004, pp. 1767–1772.

[3] P. Astrid, "Fast reduced order modeling technique for large scale LTV systems," in *Proceedings of American Control Conference 2004*, vol. 1, 2004, pp. 762– 767.

[4] A. Yousefi, B. Lohmann, J. Lienemann, and J. G. Korvink, "Nonlinear heat tranfer modelling and reduction," in *Proceedings of the 12th IEEE Mediterranean Conference on Control and Automation (MED '04)*, Kusadasi, Turkey, June 2004.

[5] L. Huisman, "Control of glass melting processes based on reduced CFD models," Ph.D. dissertation, Technishche Universiteit Eindhoven, Eindhoven (Netherlands), March 2005.

[6] L. Huisman and S. Weiland, "Identification and model predictive control of an industrial glass feeder," in *Proceedings of the 13th IFAC Symposium on System Identification (SYSID-2003)*, Rotterdam, The Netherlands, August 2003, pp. 1685–1689.

[7] S. Haykin, *Neural Networks, a comprehensive foundation*. Upper Saddle River, New jersey: Prentice Hall, 1999.

[8] L. F. Shampine and M. E. Hosea, "Analysis and implementation of TR-BDF2," *Applied Numerical Mathematics*, vol. 20, pp. 21–37, 1996.