How to Be a Gray Box: The Art of Dynamic Semi-Physical Modeling

Yacine Oussar, Gérard Dreyfus

Neural Networks, vol. 14 (2001), *invited paper*

How to Be a Gray Box:
The Art of Dynamic Semi-Physical Modeling

Yacine Oussar, Gérard Dreyfus
École Supérieure de Physique et de Chimie Industrielles de la Ville de Paris (ESPCI),
Laboratoire d'Électronique
10 rue Vauquelin
75005 PARIS

ABSTRACT

A general methodology for gray-box, or semi-physical, modeling is presented. This technique is intended to combine the best of two worlds: *knowledge-based modeling*, whereby mathematical equations are derived in order to describe a process, based on a physical (or chemical, biological, etc.) analysis, and *black-box modeling*, whereby a parameterized model is designed, whose parameters are estimated solely from measurements made on the process. The gray-box modeling technique is very valuable whenever a knowledge-based model exists, but is not fully satisfactory and cannot be improved by further analysis (or can only be improved at a very large computational cost). We describe the design methodology of a gray-box model, and illustrate it on a didactic example. We emphasize the importance of the choice of the discretization scheme used for transforming the differential equations of the knowledge-based model into a set of discrete-time recurrent equations. Finally, an application to a real, complex industrial process is presented.

I. INTRODUCTION

The traditional view of neural networks is that of "black-box models", i.e. models that are obtained from data alone, through an elaborate parameter estimation process called training. Although these methods have gained wide acceptance in industry, it is understandable that, for processes that have been investigated extensively from the point of view of physics (or chemistry, biology, etc.), there is some reluctance to forsake altogether a knowledge-based model in order to design a black-box model even though the latter is expected to be more accurate. Very frequently, a knowledge-based model is available, which is not satisfactory for the purpose of interest, but still accounts for the main features of the dynamics of the process. In such a case, it is desirable to take advantage of the existing knowledge while keeping the flexibility of parameterized models trained from data. In the present paper, we review a technique called semi-physical modeling, or knowledge-based neural modeling, which allows the model designer to incorporate, into the structure of the neural network, whatever prior knowledge is available, provided the latter is expressed as algebraic or differential equations. We present a general methodology for designing semi-physical models; we emphasize the importance of the discretization scheme used to transform differential equations arising from physics into discrete-time equations that are suitable for numerical processing. Traditionally, recurrent neural networks have been used in the framework of *explicit* discretization schemes; we show that they can also be used within the framework of *implicit* discretization schemes, which may improve the *stability* of the recurrent network model to a large extent.
The first part of the paper is devoted to the presentation of the mathematical framework of semi-physical neural modeling. The various steps of the design of a model are explained and illustrated by a didactic example. An original training algorithm is introduced, for use with models that are based on an implicit discretization scheme. In the second part of the paper, we describe an industrial application that makes use of this design strategy.

II. DYNAMIC SEMI-PHYSICAL NEURAL MODELING: MATHEMATICAL FRAMEWORK

A *knowledge-based model* is a mathematical description of the phenomena that occur in a process, based on the equations of physics and chemistry (or biology, sociology, etc.); typically, the equations involved in the model may be transport equations, equations of thermodynamics, mass conservation equations, etc. They contain parameters that have a physical meaning (e.g. activation energies, diffusion coeffi-

cients, …), and they may also contain a small number of parameters that are determined through regression from measurements.
Conversely, a *black-box model* is a parameterized description of the process, *all* parameters of which are estimated from measurements performed on the process; it does not take into account any prior knowledge on the process (or a very limited one).
A *semi-physical* (or *knowledge-based*) *neural model* may be regarded as a trade-off between a knowledge-based model and a black-box model. It may embody all the engineer's knowledge on the process (or a part thereof), and, in addition, it relies on parameterized functions, whose parameters are determined from measurements. This combination makes it possible to take into account all the phenomena that are not modeled with the required accuracy through prior knowledge. Since a larger amount of prior knowledge is used in the design of a knowledge-based neural model than in the design of a black-box model, a smaller amount of experimental data is required to estimate its parameters reliably.

II.1 Design and training of a dynamic knowledge-based neural network

II.1.1 Design principles

The design of a knowledge-based neural model requires the availability of a knowledge-based model, which is usually in the form of a set of coupled, possibly nonlinear, differential, partial differential, and algebraic, equations. We assume that this model is in standard state-space form:

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{f}\left[\boldsymbol{x}(t), \boldsymbol{u}(t)\right] \qquad \boldsymbol{y}(t) = \boldsymbol{g}\left[\boldsymbol{x}(t)\right] \tag{1}$$

where $\boldsymbol{x}$ is the vector of state variables, $\boldsymbol{y}$ is the vector of outputs, $\boldsymbol{u}$ is the vector of control inputs, and where $\boldsymbol{f}$ and $\boldsymbol{g}$ are known vector functions. This model may be unsatisfactory for various reasons: functions $\boldsymbol{f}$ and $\boldsymbol{g}$ (or some of their components) may be too inaccurate for the purpose that the model should serve, or they may involve parameters that are not estimated accurately, etc. In a black-box model, neural networks are used to approximate functions $\boldsymbol{f}$ and $\boldsymbol{g}$; they are trained from experimental data. In a semi-physical neural model, those functions that are not known accurately enough are implemented as neural models, whereas those functions, which are known reliably, are either kept under their analytic form, or implemented as a neural network with fixed weights and non-linearities.
In general, the design of a semi-physical neural model is performed in three steps:

- *step* 1*:* construction of a discrete-time semi-physical model that is derived, by an appropriate discretization scheme (see section II.2), from the knowledge-based model;
- *step* 2*:* training of the semi-physical neural model, or of specific parts thereof, from results obtained by numerical integration of the knowledge-based model; this step is generally necessary in order to obtain good initial values of the weights, to be used in step 3;
- *step* 3*:* training of the semi-physical neural model from experimental data.

This design strategy is exemplified in the next section.

II.1.2 A didactic example:

A process is described by the following second-order nonlinear model:

$$\begin{aligned} \frac{dx_1(t)}{dt} &= -\left(x_1(t) + 2\,x_2(t)\right)^2 + u(t) \\ \frac{dx_2(t)}{dt} &= 8.32\,x_1(t) \\ y(t) &= x_2(t) \end{aligned} \tag{2}$$

Figure 1 shows the process response sequences to two sequences of input steps; throughout this section, the left-hand input and output sequences will be used as the training set, and the right-hand ones as the test set.

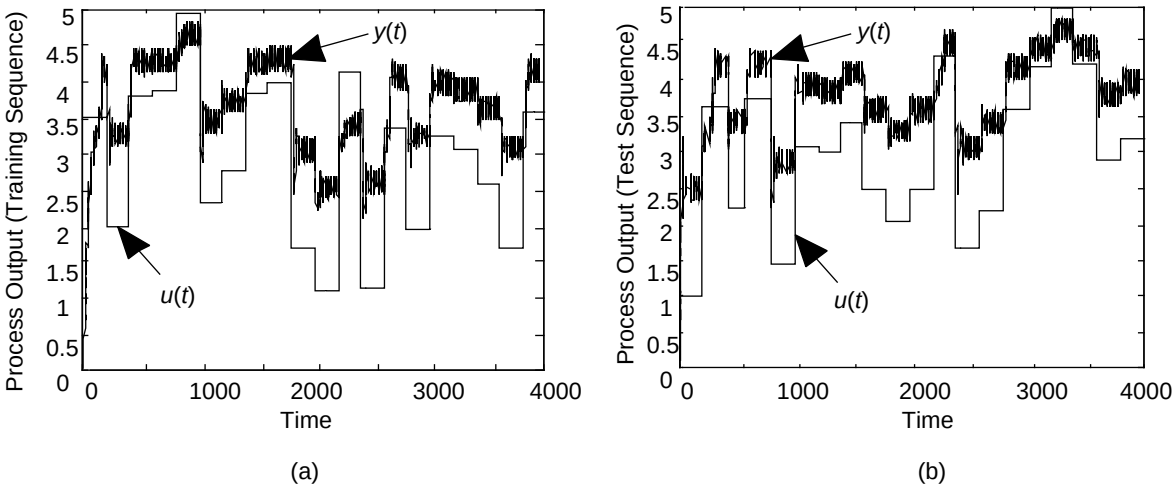

Figure 1

Process response to two input step sequences; both contain 4,000 time steps; (a): training set: (b) test set.

The results obtained by integrating the model (2) numerically are not in good agreement with experimental measurements of the output, as shown on Figure 2. The mean square modeling error is equal to 0.17, whereas the noise variance is 0.01.

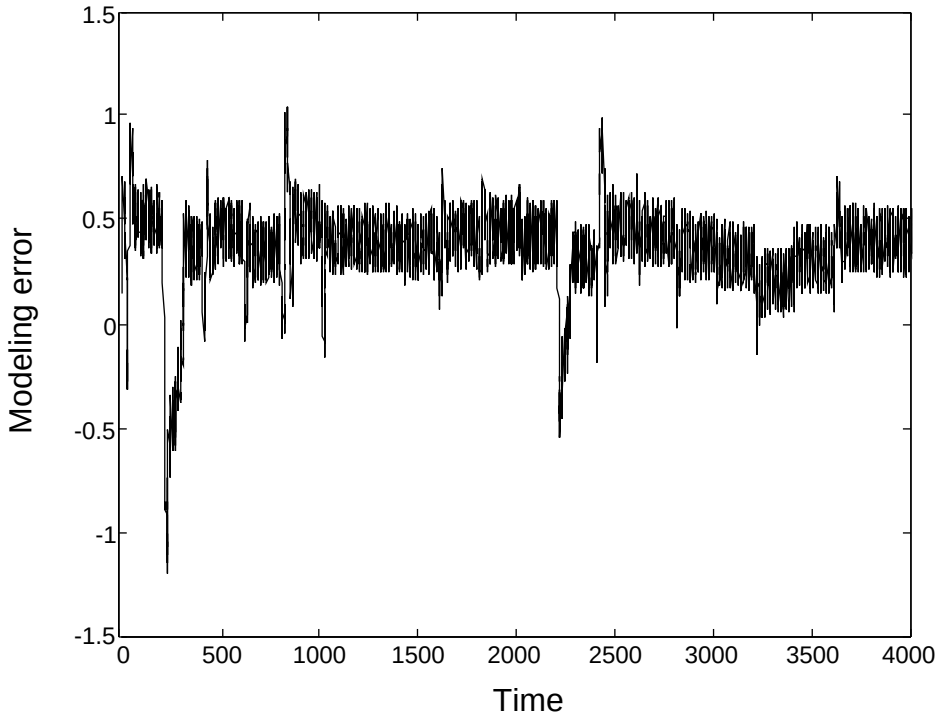

Figure 2

Modeling error of the knowledge-based model (2).

One is reasonably confident that the first state equation is valid, but there are serious doubts about the second equation:

- the parameter 8.32 may be inaccurate,
- the linear dependence is controversial,
- it is even conjectured that the right-hand side of the second equation might depend on x_2.

Therefore, in order to get a more accurate model, it may be advantageous to use a knowledge-based neural network. Actually, three different knowledge-based neural networks, of increasing complexity, may be designed in order to meet the above three criticisms. We will describe below the design of these models and the results thus obtained.

As mentioned above, the first step of the procedure consists in creating a discrete-time neural network based on the knowledge-based model. Since data is gathered with a sampling period T, the latter is a natural candidate for being the discretization

step of the equations; if Euler's explicit discretization scheme (as defined in section II.2.1) is used, the following discrete-time model is derived:

$$x_1\left[\left(k+1\right)T\right] = x_1\left(kT\right) + T\left[-\left(x_1\left(kT\right) + 2\,x_2\left(kT\right)\right)^2 + u\left(kT\right)\right] \tag{3}$$

$$x_2\left[\left(k+1\right)T\right] = x_2\left(kT\right) + T\left(8.32\,x_1\left(kT\right)\right)$$

where k is an integer. Then, the simplest knowledge-based neural model is described by the equations

$$x_1\left[\left(k+1\right)T\right] = x_1\left(kT\right) + T\left[-\left(x_1\left(kT\right) + 2\,x_2\left(kT\right)\right)^2 + u\left(kT\right)\right] \tag{4}$$

$$x_2\left[\left(k+1\right)T\right] = x_2\left(kT\right) + T\left(w\,x_1\left(kT\right)\right)$$

where w is a parameter that may be estimated by appropriate training; this model can be cast into the form of the neural network shown on Figure 3.

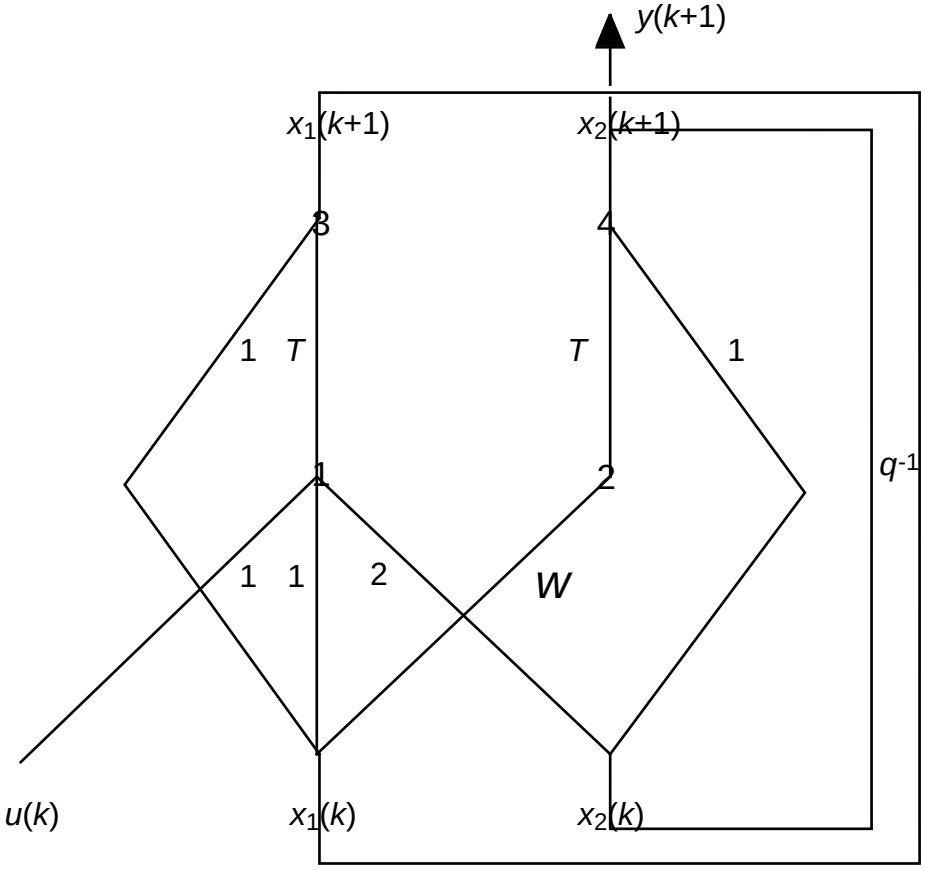

Figure 3

The simplest semi-physical model, with a single adjustable parameter w. Output of neuron 1: $o_2 = -\,[x_1(kT) + 2\,x_2(kT)]^2 + u(kT)$: output of neuron 2: $o_2 = w\,x_1(kT)$; output of neuron 3: $o_3 = T\,o_1 + x_1(kT)$; output of neuron 4: $T\,o_2 + x_2(kT)$.

For simplicity, in all the following figures, discrete time kT is simply denoted by k. q^1 is the usual symbol for a unit time delay. Neuron 1 performs a weighted sum s of x_1 and x_2, with the weights indicated on Figure 3, followed by the nonlinearity $-s^2$, and adds $u(t)$. Neuron 2 multiplies its input by the weight w. Neurons 3 and 4 just perform weighted sums. If w is taken equal to 8.32, then this network gives exactly the same results as the discrete-time knowledge-based model. If w is an adjustable weight, then its value can be computed by training the network from experimental data with any good training algorithm (evaluation of the gradient of the quadratic cost function by backpropagation through time, and gradient descent with the Levenberg-Marquardt or BFGS algorithm [Press et al., 1992]). For this training, it would be reasonable to initialize weight w to 8.32. Note that, in this very simple case, step 2 of the algorithm is bypassed.

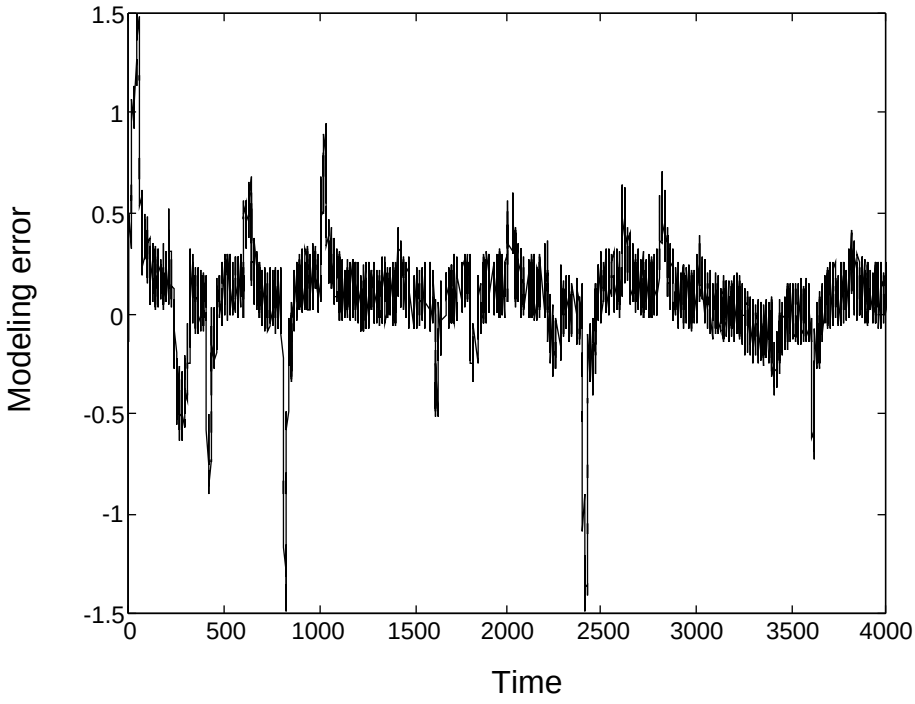

Figure 4

Modeling error of the model of Figure 3.

Figure 4 shows the modeling error with this improved model. The mean square modeling error on the test sequence is 0.08 (instead of 0.17 for the knowledge-based model); since the noise variance is 0.01, further improvement may be expected from a more elaborate model. Therefore, one considers the second level of criticism towards model (2), i.e. the fact that the linearity of neuron 2 is not appropriate; then

this neuron may be replaced by a single-layer neural network (shown on Figure 5 with three hidden neurons and 6 weights[1]).

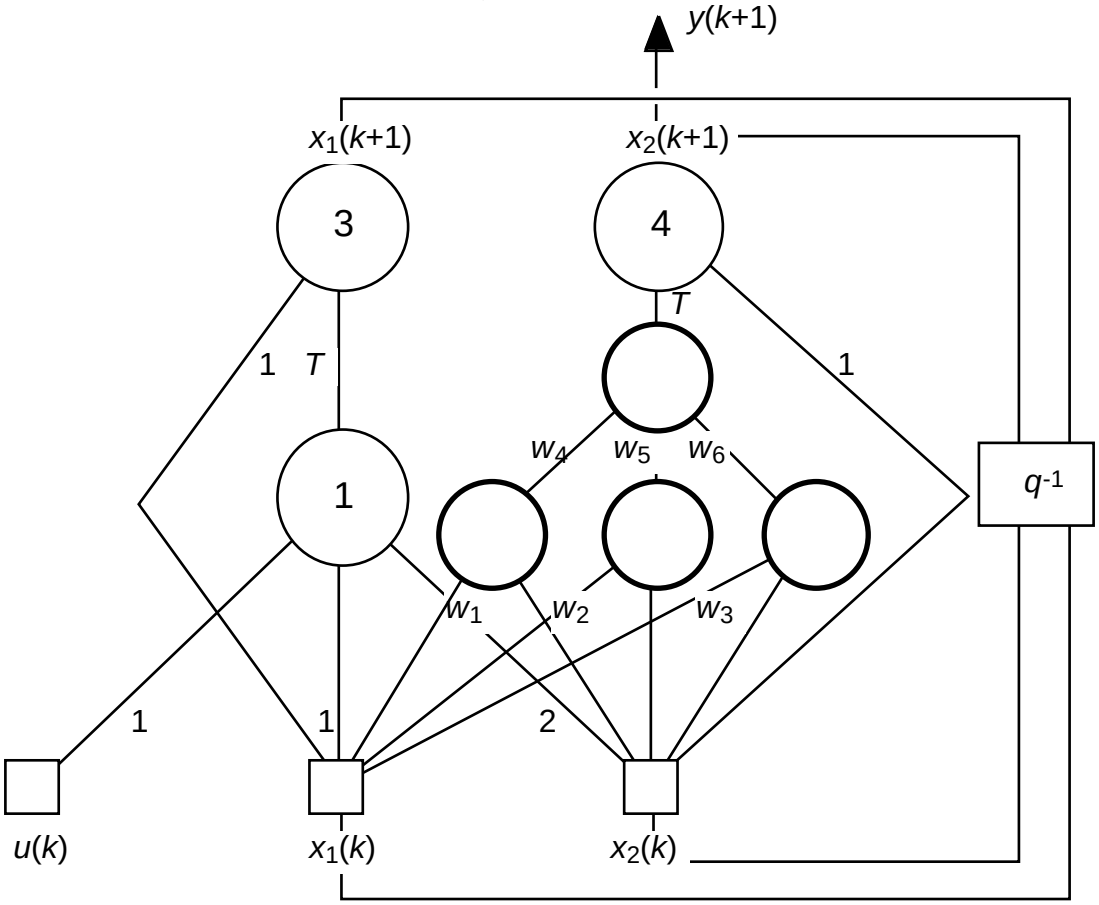

Figure 5

A more elaborate semi-physical model. The bias input is not shown. Neurons without numbers have standard sigmoid nonlinearities. Neurons 1, 3 and 4 are the same as on Figure 3.

Since $x_2(t)$ is supposed to be measurable, the feedforward neural network made of the non-numbered neurons shown on Figure 5 can be trained from data generated by the knowledge-based model (step 2 of the design procedure): although these values are known to be inaccurate, the weights resulting from this training are reasonable estimates, which are subsequently used for initializing the training of the neural network from experimental data (step 3 of the design procedure). In this case, step 2 might be bypassed, but this is not advisable: since step 3 is the training of a recurrent neural network, instabilities may occur if the weights are initialized to random values. Step 2 generates weights that are not too far from their final values, so that instabilities are unlikely to occur.

[1] Plus the weights related to the bias input, not shown on Figure 3.

Figure 6 shows the modeling error with this model, with two hidden neurons in the black-box part of the model (additional neurons generate overfitting). The mean square modeling error on the test sequence is 0.02.

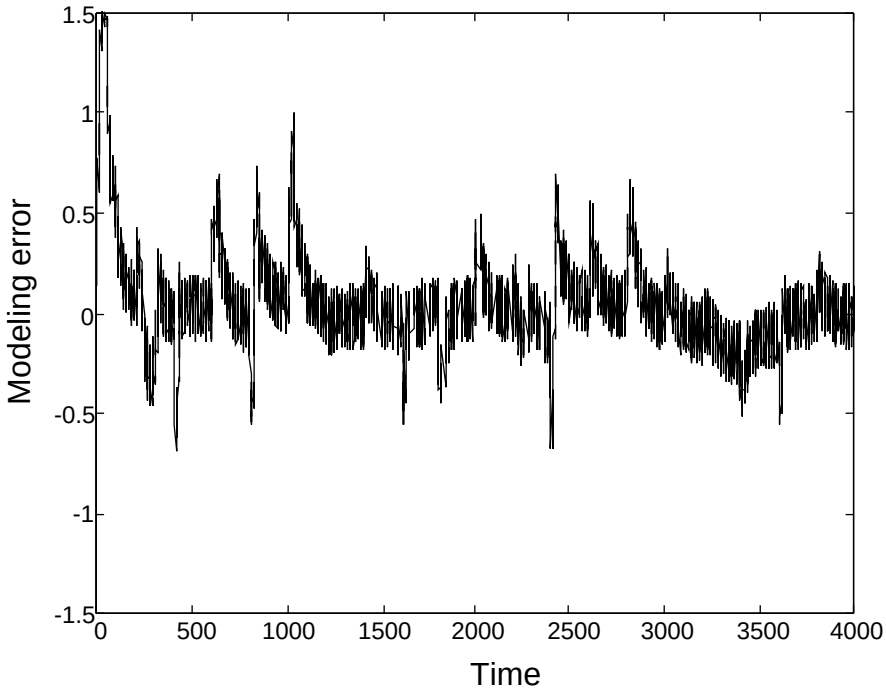

Figure 6

Modeling error of the model of Figure 5.

Finally, since the results are still unsatisfactory, the conjecture that the right-hand side of the second state equation does not depend on x_1 only, but also depends on x_2, must be taken into account. Then a third knowledge-based neural model may be designed, where the right-hand side of the second state equation is implemented as a neural network whose inputs are x_1 *and* x_2. This is shown on Figure 7 (with a feed-forward net having three hidden neurons).

Steps 2 and 3 are performed as for the previous model. Figure 8 shows the modeling error on the test sequence, with two hidden neurons in the black-box part of the model. The mean square modeling error is on the order of the noise variance, so that this model is satisfactory.

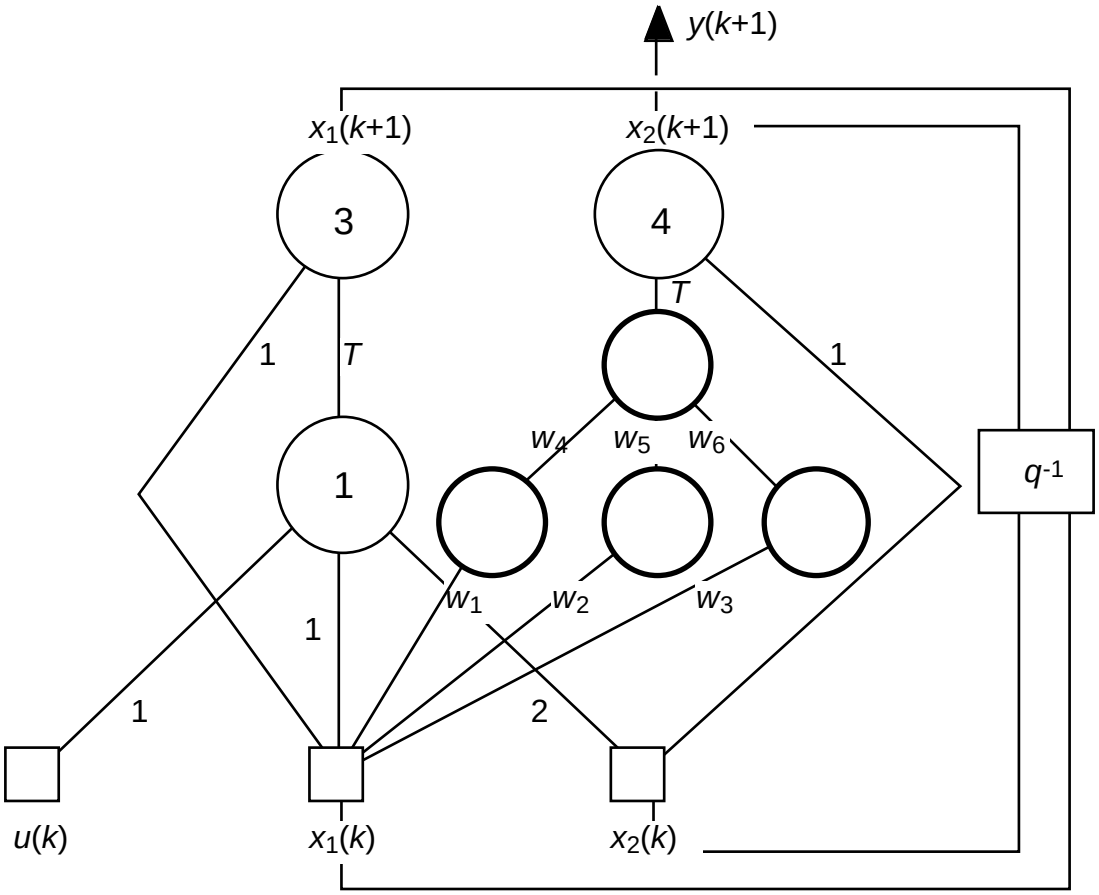

Figure 7

Still more elaborate a semi-physical model. The bias input is not shown. Neurons without numbers have standard sigmoid nonlinearities. Neurons 1, 3 and 4 are the same as on Figure 3.

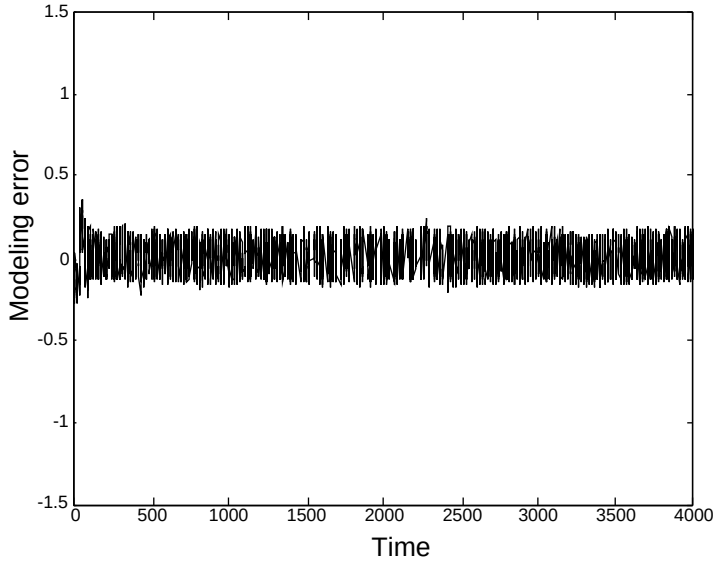

Figure 8

Modeling error of the model of Figure 7.

We have shown on this simple example the flexibility of knowledge-based neural modeling, which allows the model designer to take into account elements of prior knowledge that are considered reasonably reliable and accurate.

The first step of the design, i.e. the discretization of the knowledge-based model, is crucial because the choice of the discretization scheme may have important consequences on the stability and efficiency of the model, as well as on the training algorithm that must be used. This issue is investigated in detail in the next section.

II.2 Discretization of a knowledge-based model

As far as the design and training of a semi-physical neural model are concerned, two different classes of schemes must be considered.

II.2.1 Explicit ("forward") *vs.* implicit ("backward") discretization schemes: definitions.

Consider a first-order differential equation:

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = f\left[x(t)\right] \tag{5}$$

An *explicit* scheme discretizes it to:

$$x\left[(k+1)T\right] = \varphi\left[x(kT), T\right], \tag{6}$$

where T is the discretization step (which is frequently equal to the sampling period of the experimental data), and k is an integer. An *implicit* scheme discretizes equation (5) to:

$$x\left[(k+1)T\right] = \psi\left[x\left[(k+1)T\right], x(kT), T\right]. \tag{7}$$

More generally, consider a set of state-space equations written in vector form:

$$\frac{d\boldsymbol{x}}{dt} = f\left[\boldsymbol{x}(t), \boldsymbol{u}(t)\right]. \tag{8}$$

If an *explicit* discretization scheme is used, the discretized equations can be written under the general form:

$$K\left[\boldsymbol{x}(kT)\right]\boldsymbol{x}\left[(k+1)T\right] + \Psi\left[x(kT), u(kT), T\right] = 0, \tag{9}$$

where K is a matrix and Ψ is a vector function, whereas, if an *implicit* discretization scheme is used, the discretized equation can be written under the general form:

$$K\left[\boldsymbol{x}\left[(k+1)T\right]\right]\boldsymbol{x}\left[(k+1)T\right] + \Psi\left[x\left[(k+1)T\right], x(kT), u\left[(k+1)T\right], T\right] = 0 \ . \tag{10}$$

It will be shown in section II.2.2 that different discretizations may result in different numerical stability properties.

Examples:

Euler's *explicit* scheme consists in considering that function f is constant, equal to $f[x(kT)]$ between kT and $(k+1)T$, so that the integration of the differential equation between kT and $(k+1)T$ gives:

$$x\left[\left(k+1\right)T\right] = x\left(kT\right) + Tf\left[x\left(kT\right)\right]. \tag{11}$$

Euler's *implicit* scheme consists in considering that function f is constant, equal to $f[x((k+1)T]$ between kT and $(k+1)T$, so that the integration of the differential equation between kT and $(k+1)T$ gives:

$$x\left[\left(k+1\right)T\right] = x\left(kT\right) + hf\left[x\left[\left(k+1\right)T\right]\right]. \tag{12}$$

Similarly, Tustin's scheme consists in considering that function f varies linearly between kT and $(k+1)T$, so that the integration of the differential equation between kT and $(k+1)T$ gives:

$$x\left[\left(k+1\right)T\right] = x\left(kT\right) + \frac{T}{2}\left[f\left[x\left[\left(k+1\right)T\right]\right] + f\left[x\left(kT\right)\right]\right]. \tag{13}$$

It is therefore an implicit scheme.

Application

Consider the first knowledge-based neural model considered in section II.1; its discretization by Euler's explicit scheme has been given as relation (4). The discretization of the model by Euler's implicit scheme can be put in the following form:

$$\left[1 + Tx_1\left[\left(k+1\right)T\right] + 4\,Tx_2\left[\left(k+1\right)T\right]\right] x_1\left[\left(k+1\right)T\right] + 4\,Tx_2^2\left[\left(k+1\right)T\right] = x_1\left(kT\right) + T\,u\left[\left(k+1\right)T\right]$$

$$x_2\left[\left(k+1\right)T\right] - T\left(w\,x_1\left[\left(k+1\right)T\right]\right) = x_2\left(kT\right) \tag{14}$$

This is a specific case of relation (10), with

$$K\left[\boldsymbol{x}\left[\left(k+1\right)\right]T\right] =$$

$$\begin{pmatrix} \left[1 + Tx_1\left[\left(k+1\right)T\right] + 4\,Tx_2\left[\left(k+1\right)T\right]\right] & 4\,Tx_2\left[\left(k+1\right)T\right] \\ -Tw & 1 \end{pmatrix} \tag{15}$$

$$\Psi\left[\boldsymbol{x}\left[\left(k+1\right)T\right], \boldsymbol{x}\left(kT\right), \boldsymbol{u}\left[\left(k+1\right)T\right], T\right] = \begin{pmatrix} x_1\left(kT\right) + T\,u\left[\left(k+1\right)T\right] \\ x_2\left(kT\right) \end{pmatrix}$$

II.2.2 Explicit *vs.* implicit discretization schemes: impact on numerical stability.

The main incentive for using implicit schemes instead of explicit ones is the stability issue. For simplicity, we consider a first-order linear ordinary differential equation:

$$\frac{du(t)}{dt} = -\alpha\, u(t)\ ,\ \alpha > 0 \tag{16}$$

An Euler explicit scheme discretizes it to:

$$\frac{u\left[\left(k+1\right)T\right] - u\left(kT\right)}{T} = -\alpha\, u\left(kT\right) \tag{17}$$

which is equivalent to:

$$u\left[(k+1)T\right] = (1-\alpha T)u(kT) \tag{18}$$

Clearly, a necessary and sufficient condition for the numerical solution to be stable is:

$$T < \frac{2}{\alpha} \tag{19}$$

Therefore, the computation time is inversely proportional to α.

Consider again the simple differential equation (16). Euler's implicit scheme discretizes it to:

$$\frac{u\left[\left(k+1\right)T\right] - u\left(kT\right)}{T} = -\alpha\, u\left(kT\right) \tag{20}$$

The only difference between this equation and equation (17) is the fact that the quantity u is evaluated at time $(k+1)T$ instead of time kT. Equation (20) can be rewritten as:

$$u\left[(k+1)T\right] = \frac{1}{(1+\alpha T)}\, u(kT) \tag{21}$$

Since the denominator on the right-hand side is larger than 1, this scheme is stable irrespective of the value of α. Therefore, the choice of the discretization step is based on accuracy considerations only.

However, in contrast to the above simple example, if the equations of the model are nonlinear, they cannot be solved in closed form for the variables at time $(k+1)T$. Therefore, one has to resort, at each step of the integration, to numerical methods for solving a set of nonlinear equations. In view of the design of knowledge-based neural models, one has to find a "neural" implementation of such methods; this will be described in the next section.

II.2.3 Explicit *vs.* implicit discretization schemes: impact on neural network implementation and training.

II.2.3.1 Explicit schemes

An explicit scheme is easily implemented in the form of a recurrent neural network: since the right-hand side of the discretized equation contains quantities defined at time kT or earlier, a feedforward neural network can be used to approximate function φ (relation (6)). This has been exemplified in section II.1.

II.2.3.2 Implicit schemes

The implementation of an implicit scheme as a neural network is less straightforward than that of an explicit scheme, because the value of the state vector at time $(k+1)T$ is present on both sides of the state equation. Therefore, computing $\boldsymbol{x}[(k+1)T]$ involves solving the nonlinear equation

$$K\left[\boldsymbol{x}\left[\left(k+1\right)T\right]\right]\boldsymbol{x}\left[\left(k+1\right)T\right]+\Psi\left[x\left[\left(k+1\right)T\right],x\left(kT\right),u\left[\left(k+1\right)T\right],T\right]=0\,. \tag{22}$$

The solution of this equation can be obtained in various ways, depending on whether an analytic expression of matrix K^{-1} is available or not. This has consequences on the implementation and on the training of the model. These issues are addressed in the next subsections.

II.2.3.2.1 Matrix K can be inverted analytically

If the state vector $\boldsymbol{x}(t)$ undergoes small variations between kT and $(k+1)T$, relation (22) can be approximated as

$$K\left[\boldsymbol{x}\left(kT\right)\right]\boldsymbol{x}\left[\left(k+1\right)T\right]+\Psi\left[\boldsymbol{x}\left(kT\right),\boldsymbol{u}\left(kT\right),T\right]=0\cdot \tag{23}$$

A first approximation $x^1[(k+1)T]$ of $x[(k+1)T]$ can be obtained as

$$\boldsymbol{x}^1\left[\left(k+1\right)T\right]=-K^{-1}\left[\boldsymbol{x}\left(kT\right)\right]\Psi\left[\boldsymbol{x}\left(kT\right),\boldsymbol{u}\left(kT\right),T\right]\,. \tag{24}$$

If the dimension of matrix K is moderate, an analytic expression of its inverse can be computed. This expression can be used iteratively, so that one has, at iteration l:

$$\boldsymbol{x}^l\left[\left(k+1\right)T\right]=-K^{-1}\left[\boldsymbol{x}^{l-1}\left(kT\right)\right]\Psi\left[\boldsymbol{x}^{l-1}\left(kT\right),\boldsymbol{u}\left(kT\right),T\right]\,. \tag{25}$$

This fixed-point iteration process (also called substitution method) is terminated when it is essentially stationary, i.e. when

$$\sum_{i=1}^{N}\left|\frac{x_i^l\left[\left(k+1\right)T\right]-x_i^{l-1}\left[\left(k+1\right)T\right]}{x_i^l\left[\left(k+1\right)T\right]}\right|<\varepsilon \tag{26}$$

where ε is a constant.

Example:
Consider the model whose discretization with the implicit Euler scheme is given by relations (15). The substitution method can be applied, with:

$$K^{-1}\left[x\left(kT\right)\right]=\frac{1}{1+hx_1\left[\left(k+1\right)T\right]+4\ T\left(1+Tw\right)x_2\left[\left(k+1\right)T\right]}$$

$$\begin{pmatrix} 1 & -4\ hx_2\left[\left(k+1\right)T\right] \\ Tw & 1+Tx_1\left[\left(k+1\right)T\right]+4Tx_2\left[\left(k+1\right)T\right] \end{pmatrix} \tag{27}$$

$$\Psi\left[\boldsymbol{x}\left[\left(k+1\right)T\right],\boldsymbol{x}\left(kT\right),\boldsymbol{u}\left(kT\right),T\right]=\begin{pmatrix} x_1\left(kT\right)+Tu\left(kT\right) \\ x_2\left(kT\right) \end{pmatrix}$$

The iterative method described above allows the implementation of the model in a neural network form, hence the use of the computationally inexpensive backpropagation algorithm for the estimation of the gradient of the cost function.
As an example, consider the knowledge-based neural model whose discretization with Euler's implicit scheme is given by relation (15)**.** The resulting neural model is shown on Figure 9, featuring three successive fixed-point iterations.

Denoting by i_1, i_2 and i_3 the inputs of neuron 1 (from left to right on Figure 9), this neuron computes the quantity:

$$o_1=-\frac{i_2\left(kT\right)+Ti_1\left(kT\right)}{1+Ti_2\left[\left(k+1\right)T\right]+4T\left(1+Tw\right)i_3\left[\left(k+1\right)T\right]}+ \tag{28}$$
$$\frac{4Ti_3^2\left(kT\right)}{1+Ti_2\left[\left(k+1\right)T\right]+4\ T\left(1+Tw\right)i_3\left[\left(k+1\right)T\right]}$$

Similarly, denoting by i_1 and i_2 the inputs of neuron 2, it computes the quantity:

$$o_2=-\frac{Tw}{1+Ti_2\left[\left(k+1\right)T\right]+4T\left(1+Tw\right)i_3\left[\left(k+1\right)T\right]}\left[i_2\left(kT\right)+Ti_1\left(kT\right)\right]- \tag{29}$$
$$\frac{1+t\,i_2\left[\left(k+1\right)T\right]+4\ Ti_3\left[\left(k+1\right)T\right]}{1+Ti_2\left[\left(k+1\right)T\right]+4\ T\left(1+Tw\right)i_3\left[\left(k+1\right)T\right]}\ i_3\left(kT\right)$$

These neurons are somewhat different from the usual ones, but they are perfectly legitimate neurons, and the neural network shown on Figure 9 can be trained just as any recurrent neural network.

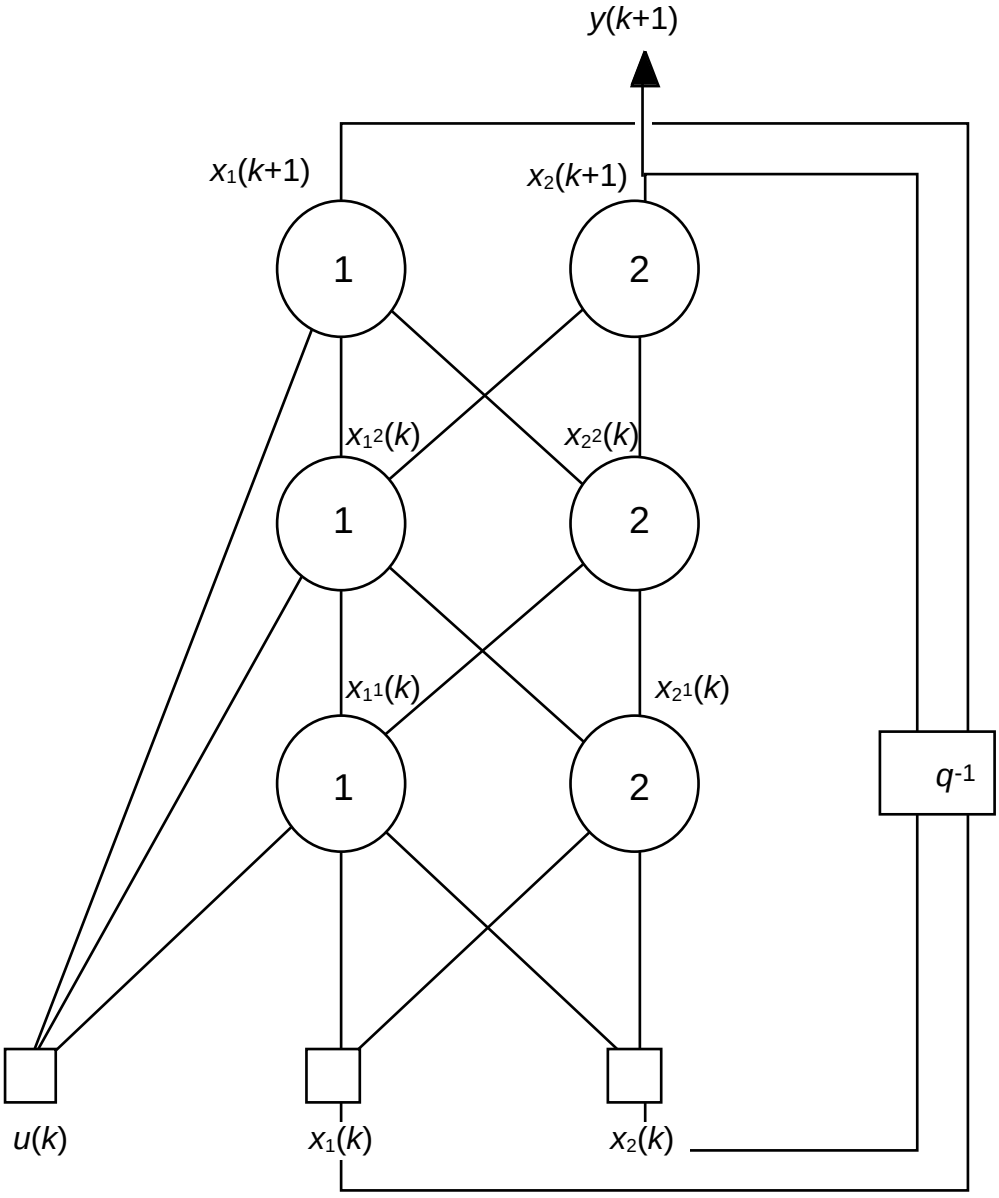

Figure 9

Neural implementation of fixed-point iteration in the substitution method.

Note that all results reported above in the didactic example (section II.1.2) were obtained through this method (explicit discretization and analytic inversion of matrix K).

II.2.3.2.2 Matrix K cannot be inverted analytically

The implementation of the substitution method in neural network form is practical if the dimension of matrix K (i.e. the number of state variables of the model) is not too large (typically smaller than 10). Otherwise, one has to resort to *numerical* matrix inversion. This requires an original method for training such models.

As mentioned in section II.2.3.2, the substitution schemes requires the iterative application of relation (23):

$$x\left[\left(k+1\right)T\right] = -K^{-1}\left[x\left(kT\right)\right]\Psi\left[x\left(kT\right), u\left(kT\right), T\right].$$

It should be remembered that the training of a dynamic neural network proceeds as follows [Nerrand et al., 1993]: the dynamic network is "unfolded in time" N times, where N is the length of the training sequence. This results in a feedforward neural network made of N identical copies of the model, which is subsequently trained by minimizing an appropriate cost function J.

In the following, we denote by θ^k the weights of copy number k, which corresponds to time step kT. The quantities $\frac{\partial J}{\partial \theta^k} = \frac{\partial J}{\partial y^k}\frac{\partial y^k}{\partial \theta^k}$ are first computed, where y^k is the output of copy k of the network. The gradient $\frac{\partial J}{\partial \theta} = \sum_k \frac{\partial J}{\partial \theta^k}$ is subsequently computed.

Therefore, it is necessary to compute the derivative of the output of a copy of the network with respect to the parameters. It has been shown in [Oussar, 1998] that the gradient of the output, computed in the forward direction (as opposed to backpropagation) is given by:

$$\frac{\partial x\left(k+1\right)}{\partial \theta} = \frac{\partial x\left(k+1\right)}{\partial x\left(k\right)}\frac{\partial x\left(k\right)}{\partial \theta} + \frac{\partial x\left(k+1\right)}{\partial \theta^{k+1}} \tag{30}$$

The application of this relation requires the computation of the matrix of the gradient of the output $x(k+1)$ of copy $k+1$ with respect to its input $x(k)$. From (23), it is given by:

$$\frac{\partial x(k+1)}{\partial x(k)} = -\frac{\partial}{\partial x(k)}\left(K^{-1}\left[x(k)\right]\Psi\left[x(k)\right]\right) \tag{31}$$

which can be rewritten as:

$$\frac{\partial x(k+1)}{\partial x(k)} = -\left(\frac{\partial}{\partial x(k)}K^{-1}\left[x(k)\right]\right)\Psi\left[x(k)\right] - K^{-1}\left[x(k)\right]\left(\frac{\partial}{\partial x(k)}\Psi\left[x(k)\right]\right) \tag{32}$$

where $\frac{\partial}{\partial x(k)}K^{-1}\left[x(k)\right]$ is a three-dimensional tensor. The latter quantity can be computed if and only if the inverse of matrix K can be computed analytically. As mentioned above, this is impractical in many cases because of the size of K.

In order to circumvent this limitation, relation (23) can be differentiated with respect to $x(k)$:

$$\frac{\partial}{\partial \boldsymbol{x}(k)}\Big(K\left[\boldsymbol{x}(k)\right]\boldsymbol{x}(k+1)\Big) = -\frac{\partial}{\partial \boldsymbol{x}(k)}\Psi\left[\boldsymbol{x}(k)\right] \tag{33}$$

After some algebra, one gets:

$$\frac{\partial \boldsymbol{x}(k+1)}{\partial \boldsymbol{x}(k)} = -K^{-1}\left[\boldsymbol{x}(k)\right]\left(\frac{\partial K\left[\boldsymbol{x}(k)\right]}{\partial \boldsymbol{x}(k)}\boldsymbol{x}(k+1) + \frac{\partial \Psi\left[\boldsymbol{x}(k)\right]}{\partial \boldsymbol{x}(k)}\right) \tag{34}$$

Therefore, the inverse of matrix K can be computed *numerically* before computing the derivatives.

Similarly, the computation of the quantity $\dfrac{\partial \boldsymbol{x}(k+1)}{\partial \theta}^{k+1}$ can be performed as follows:

$$\frac{\partial \boldsymbol{x}(k+1)}{\partial \theta}^{k+1} = -K^{-1}\left[\boldsymbol{x}(k)\right]\left(\frac{\partial K\left[\boldsymbol{x}(k)\right]}{\partial \theta}^{k+1}\boldsymbol{x}(k+1) + \frac{\partial \Psi\left[\boldsymbol{x}(k)\right]}{\partial \theta}^{k+1}\right) \tag{35}$$

The recursions are initialized with $\dfrac{\partial \boldsymbol{x}(0)}{\partial \theta} = 0$.

These relations are subsequently used in the Levenberg-Marquardt algorithm for the computation of the Hessian of the model with respect to the adjustable parameters.

II.3 Conclusion

In this tutorial section, we have explained the principles of knowledge-based neural modeling, and we have illustrated them on an academic example. We have shown how, through successive modifications, one may include various pieces of prior knowledge into the neural network. We have also shown the impact of the discretization scheme used for the knowledge-based model. We have proved that both explicit and implicit discretization schemes can be implemented as neural networks, although the implementation of the former is usually more straightforward than that of the latter; therefore, in the design phase, explicit schemes should be implemented first, and one should resort to implicit schemes only if this turns out to be mandatory.

An industrial example of knowledge-based neural modeling based on *explicit* discretization schemes can be found in [Ploix et al., 1997]. In the following section, we present an industrial application where *implicit* discretization schemes were used successfully.

III. THE KNOWLEDGE-BASED DYNAMIC MODELING OF A DRYING PROCESS.

III.1 Outline of the problem

We present a problem that was investigated in the framework of a collaboration between our group and the 3M Company.
The process to be modeled is the oven drying of a thin coating on an impermeable substrate. The coating (polymer phase) contains a single nonvolatile component - the polymer - and a single volatile component - the solvent. The solvent diffuses through the polymer and evaporates into the gas phase when reaching the surface. No diffusion takes place through the substrate. The evaporation of the solvent results in a decrease of the thickness of the coating, so that the equation to be solved is essentially the diffusion equation with a boundary condition applied at a moving boundary. In the case of organic solvents, a detailed knowledge-based model has been developed, as described in [Price et al., 1997]; when the solvent is water, then the knowledge-based model is no longer satisfactory. Since measurements are available both for organic solvents and for water, dynamic knowledge-based modeling was a natural candidate for building a model that retains most basic features of the model for solvent-based adhesives, but that adapts to water-based adhesives through training from experimental data. Experimental data consists of sequences of measurements of the residual solvent weight, with various initial values of the concentration and various oven temperatures.
The output variables of interest are:

- the solvent concentration ρ in the polymer phase,
- the temperature profile T^p in the polymer phase,
- the thickness X of the polymer phase,
- the temperature profile T^S in the substrate.

The input variables are:

- the temperature T^G in the oven,
- the initial solvent weight in the polymer phase.

Figure 10 illustrates the system.

Mass conservation is expressed as:

$$\frac{\partial \rho}{\partial t} = \frac{\partial F(x,t)}{\partial x} \tag{36}$$

where F(x, t) is the solvent flow.

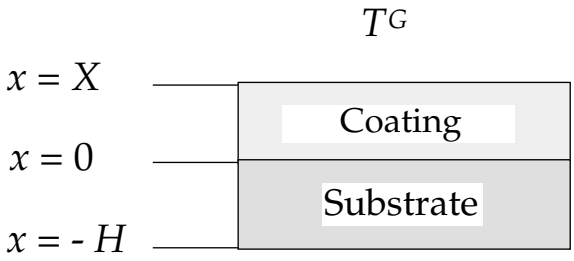

Figure 10
The physical system to be modeled.

Boundary conditions for mass conservation equation:

- The solvent flow from the coating surface into the gas phase is given by:
$$F(x,t) + \rho \frac{dX}{dt} = - k^G p^G \tag{37}$$
where p^G is the partial pressure of solvent in the gas phase and k^G is a known coefficient.
- The solvent does not diffuse through the substrate, therefore:
$F(0, t) = 0$ for all t. (38)

Volume conservation is expressed as:
$$\frac{dX}{dt} = - k^G \hat{V} p^G \tag{39}$$
where X is the polymer thickness and $\hat{V}$ is the specific volume of the solvent.

Temperature profiles:
In the knowledge-based model described in [Price et al., 1997], the variations of temperature both in time (as a function of the oven temperature) and in space (throughout the substrate and coating) are taken into account. It turns out that the typical time scale of temperature variations is very short as compared to the evaporation time. Therefore, a simplified model has been designed, which makes the simplifying assumption that the temperature in the film and in the coating is spatially uniform.

Transport equation:
The variation of flow with concentration is expressed by Fick's law:
$$F(x,t) = D \frac{\partial \rho}{\partial x} \tag{40}$$
where D is the diffusion coefficient.

Diffusion:
The dependence, on temperature and solvent concentration, of the diffusion coefficient D of the solvent in the polymer, is given by the free-volume theory[2]:

$$D = Q\,\boldsymbol{D_0} \exp\left[-\frac{\boldsymbol{E}}{RT^P}\right] \exp\left[-\Phi\right] \tag{41}$$

where E is an activation energy, and where Φ is a function, known analytically, of the concentration of the solvent, involving free-volume parameters for the solvent and the polymer which are usually estimated through regression.

Driving force for drying:
Clearly, the driving force of the process is the partial pressure of the solvent in the gas phase (present in equations (37) and (39)); it was derived from the Flory-Huggins theory, and the solvent vapor pressure was given by the Antoine equation:

$$p^G = P\,\phi_1^P \exp\left[\phi_2^P + \chi\left(\phi_2^P\right)^2\right] \tag{42}$$

where p^G is the solvent partial pressure in the gas, $\phi_1^P = \frac{\rho}{\rho_0}$, $\phi_1^P + \phi_2^P = 1$, ρ_0 is the inverse specific volume, and χ is the Flory-Huggins parameter. P is the solvent vapor pressure given by the Antoine equation:

$$\log_{10} P = A - \frac{B}{T^P(X,t) + C} \tag{43}$$

where A, B and C are known parameters.

III.2 Design of a knowledge-based neural model of the process

The first step of the design of the knowledge-based neural model is the choice of the equations that must still be present in the neural model, and of those that correspond to assumptions that may be relaxed in the neural model.
Clearly, the equations expressing

- mass conservation
- boundary conditions
- volume conservation
- temperature profiles

are valid irrespective of the nature of the solvent, and therefore must be retained in the knowledge-based neural model.

---

[2] Boldface symbols denote quantities that are unknown, and are estimated by nonlinear regression, from measured data.

By contrast, the validity of the last three equations (Fick's law, temperature and concentration dependence of the diffusion coefficient predicted by free-volume theory, Flory-Huggins theory for the driving force) may be questioned in the case of water-based adhesives. Therefore, black-box neural networks, within a knowledge-based neural model, may replace all or some of them.
As described above, various models of different complexity were designed. The best results were obtained with a network in which the temperature and concentration dependence of the diffusion coefficient was replaced by a neural network, i.e. in which free-volume theory was dropped.
An explicit scheme was used for space discretization, with a geometrically decreasing step because of the rapid variation of the concentrations near the free surface. An implicit time discretization was found mandatory, because the diffusion coefficient varies over several orders of magnitude throughout the experiment.
The overall structure of the model, for training, is shown on Figure 11. The concentration (in appropriately reduced variables) in slice j at time n is denoted by c_j^n. Reduced temperature at time n is denoted by T^*_n; reduced thickness at time n is denoted by X^*_n. At each space grid point, a neural network NN computes the local diffusion coefficient; the shared weight technique is used, since the neural networks are the same at each grid point. This allows the computation of all elements of matrix K and vector Ψ. The inverse K^{-1} is computed, and relation (23) is implemented by multiplying K^{-1} by Ψ, thereby obtaining the values of the state variables at time $n+1$. The upper part of the network implements integration of the concentration over the thickness of the sample, with space grid steps $\Delta\eta_j$, j=1 to M, in order to get the measured quantity, i.e. the residual solvent weight at time $n+1$.

The discretization of the state-space equations by an implicit scheme was found mandatory because the diffusion coefficient varies over several orders of magnitude from boundary to boundary, so that numerical stability is a critical issue. Since the state-space equation were discretized by an implicit scheme, training through numerical matrix inversion was performed as described in section II.2.3.2.2.
In order to validate the method and the structure of the network, a neural network was trained from data obtained with solvent-based adhesives (for which a satisfactory knowledge-based model exists. A typical result is shown on Figure 12: the prediction is very good (within the accuracy of the measurements).
Figure 13 shows typical results obtained with water-based adhesives. As might be expected, the accuracy of the prediction is not quite as good as that obtained with solvent-based adhesives, but it is nonetheless very satisfactory.

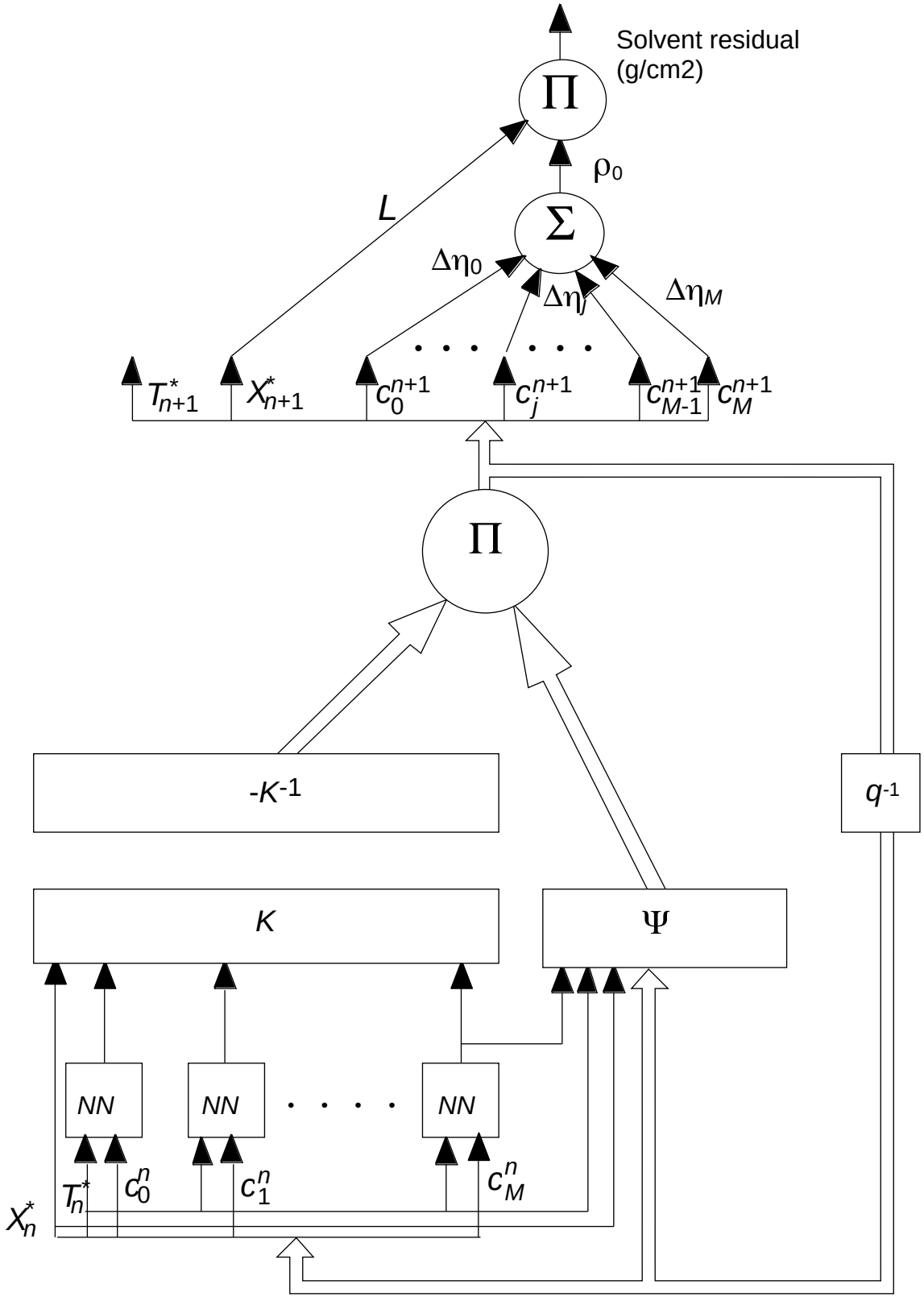
Solvent residual
(g/cm2)
Π
ρ0
Σ
L
Δη0
Δηj
ΔηM
T*n+1
X*n+1
c0n+1
cjn+1
cM-1n+1
cMn+1
Π
-K-1
q-1
K
Ψ
NN
NN
NN
T*n
X*n
c0n
c1n
cMn

Figure 11

Overall structure of the semi-physical model of a drying process.

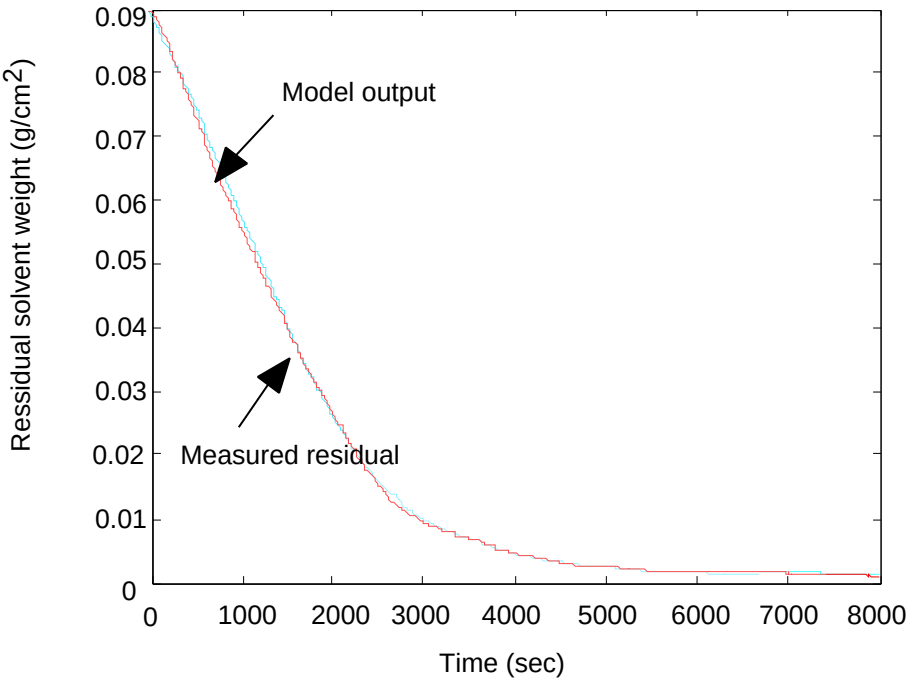

Figure 12

Results for an organic solvent-based adhesive.

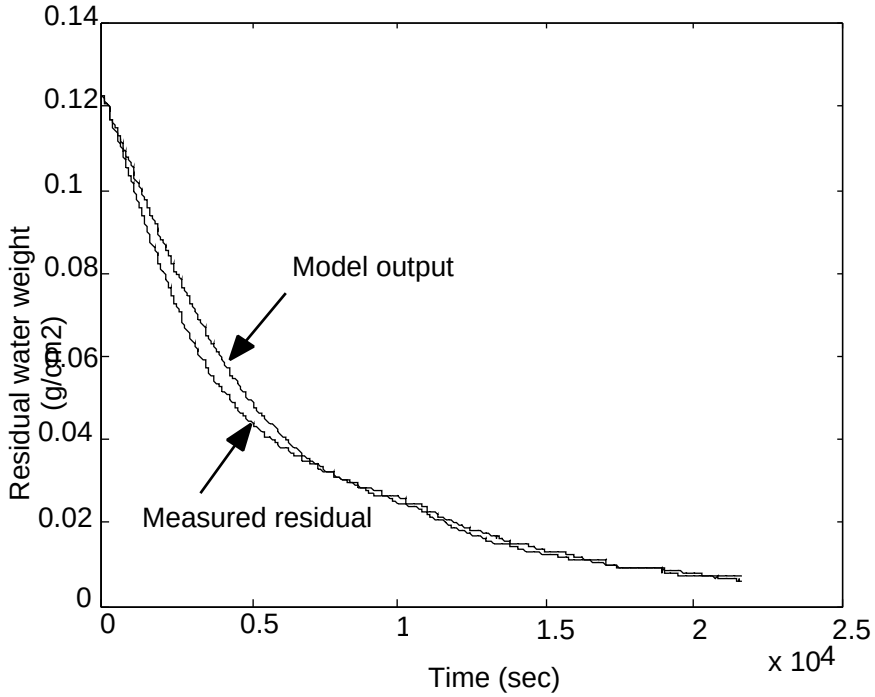

Figure 13

Results for a water-based adhesive.

IV. CONCLUSION

We have shown that dynamic semi-physical (knowledge-based) neural modeling can be a very powerful strategy for the design of models that combine the best of two worlds: the legibility of knowledge-based models and the flexibility of training from experimental data. It allows the integration of the mathematical equations derived from a knowledge-based model into the structure of the neural network. A very important issue, namely the stability of the recurrent neural network model, has been investigated; it has been shown how the good numerical stability properties of implicit discretization schemes can be taken advantage of, and an original algorithm, specific to recurrent networks based on implicit discretization, has been derived.

Because of its hybrid nature, this design strategy should gain acceptance in the field of engineering; an industrial application has been described, that takes full advantage of semi-physical modeling.

LITERATURE REFERENCES

Press, W. H., Teukolsky S.A., Vetterling, W.T., & Flannery, B.P. (1992). *Numerical Recipes in C : The Art of Scientific Computing*. Second Edition, Cambridge university Press.

Ploix, J.L. & Dreyfus, G. (1997). Early fault detection in a distillation column: an industrial application of knowledge-based neural modelling. In B. Kappen & S. Gielen (Eds), *Neural Networks: Best Practice in Europe* (pp. 21-31). World Scientific.

Price, P.E., Wang, S., & Romdhane, I.H. (1997). Extracting Effective Diffusion Parameters from Drying Experiments. *AIChE Journal,* 43, 1925-1934.

Nerrand, O., Roussel-Ragot, P., Personnaz, L., Dreyfus, G., & Marcos, S. (1993). Neural Networks and Non-linear Adaptive Filtering: Unifying Concepts and New Algorithms. *Neural Computation*, 5, 165-197.

Oussar, Y. (1998). *Réseaux d'ondelettes et réseaux de neurones pour la modélisation statique et dynamique de processus*. Thèse de Doctorat de l'Université Pierre et Marie Curie - Paris VI. Available from http://www.neurones.espci.fr

ACKNOWLEDGEMENTS

The authors are very grateful to Dr. Romdhane, Dr. Stoppiglia and Dr. Kinoghlu, of 3M, for many fruitful, stimulating and enlightening discussions.