Noname manuscript No.
(will be inserted by the editor)

Smooth Statistical Modeling of Bivariate Non-monotonic Data by a Three-Stage LUT Neural System

Simone Fiori · Nicola Fioranelli

the date of receipt and acceptance should be inserted later

Abstract The present paper introduces a new statistical data-modeling algorithm based on artificial neural systems. This procedure allows abstracting from datasets by working on their probability density functions. The proposed method strives to capture the overall structure of the analyzed data, exhibits competitive computational runtimes and may be applied to non-monotonic real-world data (building on a previously developed isotonic neural modeling algorithm). An outstanding feature of the proposed method is the ability to return a smoother model compared to other modeling algorithms. Smooth models could have applications in the fields of engineering and computer science. In fact, the present research was motivated by an image-contour resampling problem that arises in shape analysis. The features of the proposed algorithm are illustrated and compared to the features of existing algorithms by means of numerical tests on shape resampling.

Cite as: ***S. Fiori and N. Fioranelli, Smooth statistical modeling of bivariate non-monotonic data by a three-stage LUT neural system, Neural Computing and Applications (Springer), Vol. 30, No. 4, pp. 1353 – 1368, 2018 (DOI: 10.1007/s00521-017-3215-1)***

S. Fiori
Dipartimento di Ingegneria dell'Informazione, Facoltà di Ingegneria, Università Politecnica delle Marche, Via Brecce Bianche, I-60131 Ancona (Italy)
Corresponding author. E-mail: s.fiori@univpm.it

N. Fioranelli
School of Information and Automation Engineering, Università Politecnica delle Marche, Via Brecce Bianche, I-60131 Ancona (Italy)

Keywords Look-up table neural networks · Smooth data modeling · Isotonic data modeling · Non-monotonic data modeling

1 Introduction

A large number of data processing systems deal with non-linear data modeling. In particular, bivariate data modeling aims at (approximately) inferring the complex relationship between a single independent variable $X \in \mathbb{R}$ and a single dependent variable $Y \in \mathbb{R}$ [30]. In the present contibution, we are dealing with *bivariate regression* as an instance of bivariate data modeling. A good deal of research has been devoted to modeling such non-linear relationships by neural networks and systems [31]. A number of applications in engineering and computer science deal with nonlinear data modeling by artificial intelligence techniques. Examples of these research efforts published over the past few years are a work on modeling of seed spacing uniformity of a pneumatic planter [1], a work on neural network modeling of deformations in masonry structures [6], a work on predicting heat fluxes and evapo-transpiration [8], a work on crop yield modeling and forecasting [17], a work on credit rating modeling [18], a research on water quality prediction [25], a research on the estimation of melting points of fatty acids [27], a work on surface roughness prediction [28], and a research on forecasingt electricity prices [29].

Traditional nonlinear modeling techniques are based on a non-linear model endowed with a number of tunable parameters that are adapted in order to minimize the misfit between the predicted model's outcome and the actual observations. Neural-network-based nonlinear regression is based on the same principle, except

that the non-linear functional model is not specifically tailored to the underlying data but is a generic neural network structure, such as a Multi-Layer Perceptron (MLP, see, for example, the comparison presented in [37] between linear, polynomial, log reciprocal, von Bertalanffy, Gompertz, logistic and exponential models and an artificial neural network method) or a General Regression Neural Network (GRNN) [32] (see also [33] for a comparison between neural networks and decision trees in nonlinear regression).

A specific modeling problem arises when the underlying relationship between the dependent variable and the independent variable is known to be *monotonic*. Such problem possesses a specific solution known as *isotonic modeling* [2]. In particular, the first author proposed in the contribution [11] a *statistical isotonic modeling* algorithm based on Look-up Table Neural Systems (LUTs). LUT-based artificial neural networks are often invoked in the literature when it is necessary to implement non-linear activation functions efficiently (see, for example, the application to real-time neural-network inversion described in [5]). A LUT-based neural network is a weightless, flexible neural model whose non-linear activation function is not fixed a priori (e.g., it is not fixed as a sigmoidal activation function). The non-linear activation function is rather learnt from the training set by means of a learning rule tailored to the task at hand. The non-linear activation function is represented as a look-up table whose entries are modified according to a learning rule. A bivariate N-size look-up table is essentially a set of pairs $\{(x_i, y_i) \mid i = 1, 2, \ldots, N\}$ together with a set of (simple) mathematical operators. For a general survey, readers might consult, e.g., the book [19], while specific surveys on LUT-based neural networks in the context of signal processing and data processing may be found within the neural-network and neural-computing papers [10,11,22]. An interesting application of LUTs to implement efficient artificial neural networks may be found in the academic thesis [35].

A limiting drawback of the modeling method discussed in [11] is that it is capable of capturing only monotonic relations. Non-monotonic relationships between two variables are encountered when the dependent variable is not consistently increasing and never decreasing or consistently decreasing and never increasing in value but takes a non-linear trend. An example of non-monotonic behavior, with important consequences on toxicological risk assessment, may be found in the reactions of a complex biological system to a toxicant, where biphasic dose-effect relationships can be observed, showing a decrease at low doses followed by an increase at high doses [3]. Another well-known example of a real-world system exhibiting a non-monotonic relationship between two variables is the tunnel diode (or Esaki diode), whose voltage-current characteristics look markedly nonlinear and non-monotonic [7].

Isotonic modeling has been generalized in different ways in the recent past (see, for instance, the contributions [4,14,23,34,36]). The present paper aims at extending the statistical isotonic modeling method presented in the paper [11] to model non-monotonic datasets.

The present research was motivated, in particular, by an image-contour resampling problem that arises in the pre-processing of contours in shape analysis, as explained in [20]. Resampling a contour for image classification purposes is a specific data-processing problem where it is more essential for the resampled contour to be smooth than to be precisely close to the original contour, for at least two reasons. A first reason is that a shape-contour dataset arising from an acquired image may be noisy, and a very precise model will incorporate the unwanted noise, while a smooth model will filter out the noise components. A second reason is that, as far as the estimation of the local curvature of a shape-contour is concerned, smooth resampling is necessary to allow computing first-order derivatives with little hassles (for a numerical example of curvature estimation, see the *Experiment 2* in Subsection 4.2).

The present paper is organized as follows. Section 2 summarizes previous contributions by the first author and coworkers, published in the papers [11,14], that the present endeavor is based on. Section 3 explains the transformation principle used to turn a non-monotonic dataset into a monotonic one and illustrates the whole non-linear regression procedure by pseudo-coding. Section 4 illustrates the capabilities of the devised neural regression technique by numerical tests on a shape-resampling problem. Section 5 concludes the manuscript.

2 Previous contributions

The first author contributed to the problem of statistical isotonic regression by a method published in the paper [11], which is summarized, for the benefit of the readers, in the Subsection 2.1. Such method was later extended to statistical non-isotonic regression in [14]. The present paper also deals with statistical non-isotonic regression by means of a novel algorithm, whose computational complexity will be compared (see Subsection 4.1) to the computational burden exhibited by the neural algorithms proposed in [14], which are summarized in Subsection 2.2.

2.1 Statistical isotonic regression [11]

Let us assume that we want to model a bivariate dataset $\mathbb{D} = \{(x_i, y_i) \in \mathbb{R}^2 \mid i = 1, 2, 3, \ldots, N\}$ and that the relationship between the samples x_i and the samples y_i is monotonic. (For example, if the relationship between the samples x_i and the samples y_i is monotonically increasing, we know that $x_j > x_i$ implies that $y_j > y_i$). Let us denote the model underlying the data by $f : \mathbb{R} \to \mathbb{R}$ and take this model to be monotonically increasing (namely, $df/dx > 0$) or monotonically decreasing (namely, $df/dx < 0$).

A first noteworthy feature of the algorithm proposed in [11] is that such method does not insist on the data to be modeled but on their statistical distributions. Namely, denoting by x_i the samples of the independent variable and by y_i the samples of the dependent variable, which are realizations of the random variables X and Y, respectively, the algorithm proposed in [11] does not make direct use of the samples x_i, y_i but of the probability density functions $p_X(x)$ and $p_Y(y)$, that are estimated through the method of occurrence histograms. As a consequence, while traditional data-modeling techniques try to fit a pre-defined model to the available data, the *statistical* modeling technique matches their statistical distributions by means of a nonlinear transform: a model of the data (see, for example, the large ensemble of numerical experiments on real-world, noisy data sets, including nuisance variables, presented in [12].)

A second important feature of the statistical isotonic modeling method proposed in [11] is that *the model may be expressed in closed form*. Namely, upon defining the following cumulative distribution functions:

$$P_X(x) = \int_{-\infty}^{x} p_X(x)dx, \; P_Y(y) = \int_{-\infty}^{y} p_Y(y)dy, \qquad (1)$$

the statistical isotonic model of the dataset $\mathbb{D}$ may be written as:

$$f(x) = \begin{cases} P_Y^{-1}(P_X(x)) & \text{monot. increasing relation,} \\ P_Y^{-1}(1 - P_X(x)) & \text{monot. decreasing relation,} \end{cases} \qquad (2)$$

where the symbol P_Y^{-1} denotes the inverse of the function P_Y, that exists as long as $p_Y(y) \neq 0$.

A third, distinguishing feature of the statistical isotonic modeling method being recalled is that the involved nonlinear functions are represented through Look-up Table neural systems, which make the required numerical operations (including inverting the cumulative distribution function P_Y) be computationally light and the whole modeling procedure fast to execute. In addition, since such modeling method does not involve any parameter to tune nor any basis functions to be selected in advance, the produced bivariate model is flexible and may follow the shape of a broad class of underlying relationships, which may present, for instance, abrupt changes.

For more technical details, readers might consult the paper [11].

2.2 Non-isotonic statistical regression [14]

Let us consider, now, datasets that do not present a monotonic trend (namely, even if $x_j > x_i$, it could happen that $y_j > y_i$ or $y_j < y_i$ or even $y_j = y_i$). Consequently, the model f may take any local curvature, namely, its first-order derivative $df/dx \gtreqless 0$. For such kinds of datasets, it is impossible to apply the neural statistical modeling method proposed in [11], which is based on the assumption that the model exhibits a strict monotonic (either increasing or decreasing) shape.

In the recent publication [14], an idea was explored that allows a neural isotonic modeling procedure to be applied to non-monotonic datasets, based on a three-stage neural system. The key idea is that non-monotonic data can be transformed to monotonic data by an invertible transform. After such transformation, neural isotonic regression may be applied to the monotonized training set and the regression result may be transformed back to its non-monotonic form. The non-linear transform, that makes non-monotonic data approximately monotonic, will be implemented by a learnable LUT neural system.

The Figure 1 illustrates this neural system that is based on the following sequence of operations: 1) modify the original non-monotonic dataset through a *non-linear, invertible transformation* that makes it monotonic. Such non-linear transform is to be learned from the original data; 2) apply the neural statistical isotonic procedure to the 'monotonized' dataset; 3) bring back the model to its original domain by applying the inverse non-linear transform.

As a first stage, in order to make an isotonic modeling procedure be able to cope with such a non-monotonic relationship, it is necessary to transform the model f into a monotonic model $h = \Gamma(f)$ by means of a transformation Γ such that $dh/dx > 0$, strictly. In the earlier contribution [14], a multiplicative non-linear transform (implemented by a learnable LUT neural network) was invoked, which takes the form

$$\Gamma(f(x)) = g(x)[f(x) + \kappa],$$

where the function $g : \mathbb{R} \to \mathbb{R}$ and the constant $\kappa \in \mathbb{R}$ need to be learned from the dataset $\mathbb{D}$. Such nonlin-

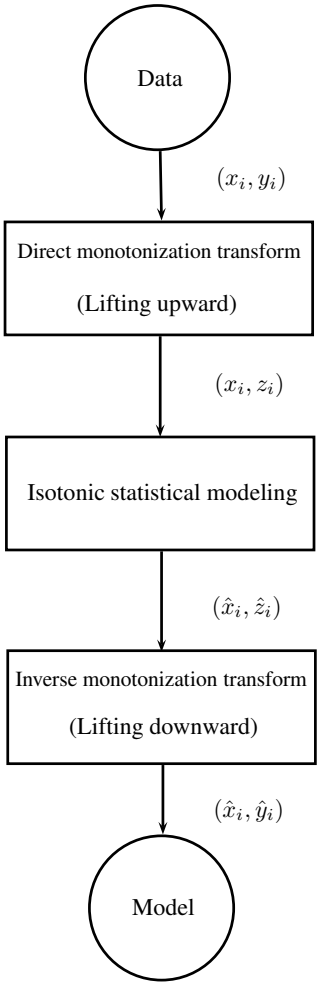

Fig. 1 Three-stage neural system that allows modeling a non-monotonic dataset by means of an isotonic data-modeling algorithm. The three stages, represented by rectangular boxes, are all non-linear in the formulation proposed in [14], while they are, respectively, linear, non-linear and linear in the present proposal (the linear stages are denoted in parentheses and the input/output data-samples are indicated in each stage).

ear transform works effectively but, being dependent of the data, it might be cumbersome to learn and its performances vary from dataset to dataset (see the large ensemble of numerical experiments presented in [14]). Because of the non-linearity of the data-transform neural system, such three-stage procedure may be defined as 'nonlinear-nonlinear-nonlinear'.

The transfer function of the neural data-monotonization system reads $h(x) = g(x)\bar{f}(x)$, where $\bar{f}(x)$ denotes the model-function $f(x)$ lifted upward in such a way that $\bar{f}(x) > 0$ for every value of the variable x. The neural transfer function g is termed *multiplicative transform function*. The monotonicity condition on the function $h(x)$ can be written as:

$$\frac{dh}{dx} = \frac{dg}{dx}\bar{f} + g\frac{d\bar{f}}{dx} > 0, \tag{3}$$

and is enforced by means of an appropriate learning rule for the neural system. The paper [14] proposed three learning rules to infer a transfer function, which we summarize below to give a general review:

– *Learning rule 1*: It is based on data pre-smoothing by a Gaussian kernel denoted by $k : \mathbb{R} \to \mathbb{R}$, on the pre-estimation of the model and on the learning of the multiplicative transform function. In order to write a rule that allows learning the multiplicative transform function, the range of the variable x is subdivided into n equally-spaced sections of width $\Delta = (x_{\max} - x_{\min})/n$, and a new set of n grid points $\{\zeta_0, \zeta_1, \ldots, \zeta_\lambda, \ldots, \zeta_n\}$ is generated, where $\zeta_\lambda = x_0 + \lambda\Delta$, with $\lambda = 1, \ldots, n$. Each value $k(\zeta_\lambda)$ is calculated by cubic spline interpolation so that the derivative function $k'(x) = dk/dx$ can be numerically approximated over the grid-points ζ_λ. Each value $g(\zeta_\lambda)$ is calculated by estimating $g'(\zeta_\lambda)$ over the grid points by:

$$g(\zeta_\lambda) = \Delta g'(\zeta_{\lambda-1}) + g(\zeta_{\lambda-1}), \tag{4}$$

$$g'(\zeta_\lambda) = -\frac{k'(\zeta_\lambda)}{k(\zeta_\lambda)} g(\zeta_\lambda) + \epsilon_1, \tag{5}$$

where the boundary condition is set to $g(x_0) = 1$ and $\epsilon_1 > 0$ is a parameter of the learning rule. The values $g(x_i)$ of the neural transfer function are calculated from the values $g(\zeta_\lambda)$ by interpolation. The original dataset $\mathbb{D} = \{(x_i, y_i) \mid i = 1, 2, \ldots, N\}$ is transformed to be a monotonically increasing dataset $\{(x_i, z_i) \mid i = 1, 2, \ldots, N\}$, with $z_i = g(x_i)\,\bar{y}_i$, where $\bar{y}_i$ denotes the lifted-upward version of the data y_i such that all $\bar{y}_i > 0$. Isotonic regression is then applied to the transformed dataset made of the pairs (x_i, z_i).

– *Learning rule 2*: It allows learning a neural transfer function by a multiplicative update rule. As opposed to the previous learning method, the multiplicative transform function g is learnt directly from the training set without any smoothing nor pre-estimation. The first step consists again in shifting upwards the training set in order to ensure that the y-samples be positive-valued. Let $\bar{y}_i = y_i + \gamma$, such that $\bar{y}_i > 0$ holds for every value of the index i and let us set $\bar{f}(x_i) = \bar{y}_i$. Let us set, for the sake of notation conciseness, $g_i = g(x_i)$. To learn the value g_i of the look-up table that represents the non-linear transfer function of the multiplicative transform neural system, let us set $g_1 = 1$, and let the

system learn the next values through the recursive rule:

$$g_i = M \frac{g_{i-1}\bar{y}_i}{2\bar{y}_i - \bar{y}_{i-1}}, \tag{6}$$

with $M > 1$. The above neural learning rule is well-defined as long as the $\bar{y}$-data satisfy the condition $2\bar{y}_i - \bar{y}_{i-1} \neq 0$ for every value of the index i, which may be ensured by shifting the data upward some further. The original dataset $\mathbb{D}$ is transformed to be a monotonically increasing dataset $\{(x_i, z_i) \mid i = 1,\ 2,\ \ldots,\ N\}$, with $z_i = g_i\,\bar{y}_i$. Isotonic regression is then applied to the transformed dataset made of the pairs (x_i, z_i).

– *Learning rule 3*: It allows learning a neural transfer function by an additive rule. Instead of the multiplicative update rule (6), the following additive learning rule was proposed:

$$g_i = \frac{g_{i-1}\bar{y}_i}{2\bar{y}_i - \bar{y}_{i-1}} + \epsilon_2. \tag{7}$$

The constant $\epsilon_2 > 0$ is set to a low value. Again, the original dataset $\mathbb{D}$ is transformed to be a monotonically increasing dataset $\{(x_i, z_i) \mid i = 1,\ 2,\ \ldots,\ N\}$, with $z_i = g_i\,\bar{y}_i$, and isotonic regression is applied to the transformed dataset (x_i, z_i).

For further implementation details about the above three learning rules and for a detailed discussion about their features and drawbacks, we refers the readers to the paper [14].

As an interesting introduction to the topic of data-trasformation, we mention the contribution [24]. Such paper explains that transforming a dataset is traditionally considered difficult because of the need to guess a non-linear transformation of the data out of experience. Moreover, such paper discusses how nonlinear modeling differs from traditional approaches such as polynomial regression and cubic spline. We would also like to mention a review of a closely-related topic, namely, *monotonic classification* [21], that makes use of data monotonization, although the meaning of 'monotonization', and the way to achieve it, are quite different from those discussed in the present contribution.

3 Novel data transformation principle and implementation details

Compared to the previous contribution [14], the present paper explores a much simpler data transformation rule, namely:

$$h(x) = \Gamma(f(x)) = f(x) + c\,x, \tag{8}$$

with $c \in \mathbb{R}$ denoting an appropriately chosen constant. The requirement that $h'(x) > 0$ can be met under the assumption that *the first order derivative of the model f be absolutely bounded*. In fact, to ensure a monotonic behavior of the model, it is sufficient to take:

$$c > \max_x \left\{ -\frac{df(x)}{dx} \right\}. \tag{9}$$

Thanks to the linearity of the data-transform (8), the resulting three-stage modeling procedure may be defined as 'linear-nonlinear-linear'. In real-world applications, the model f is unknown, hence the exact relationship (9) needs to be implemented data-wise, namely, the dataset $\mathbb{D}$, made of N sample-pairs, needs to be transformed into the dataset $\mathbb{D}^\ell = \{(x_i, z_i) \in \mathbb{R}^2 \mid i = 1, 2, 3, \ldots, N\}$, with

$$z_i = y_i + c\,x_i, \text{ with } c > \max_i \left\{ \frac{y_{i-1} - y_i}{x_i - x_{i-1}} \right\}, \tag{10}$$

where the superscript ℓ stands for 'lifted'. Without loss of generality, it is assumed that all the samples x_i are distinct from each other. If, however, there exist multiple records with an identical x_i-value, low-valued random displacements may be added to the x-value of those records, in such a way that those records may be sorted [14].

Once the isotonic statistical modeling procedure has returned a (lifted) neural model $\mathbb{M}^\ell = \{(\hat{x}_i, \hat{z}_i) \in \mathbb{R}^2 \mid i = 1, 2, 3, \ldots, R\}$ over a grid of R points-of-interest $\hat{x}_i$, the obtained model will be shifted downwards to its original domain by the rule

$$\hat{y}_i = \hat{z}_i - c\,\hat{x}_i, \tag{11}$$

to give the LUT-based neural model $\mathbb{M} = \{(\hat{x}_i, \hat{y}_i) \in \mathbb{R}^2 \mid i = 1, 2, 3, \ldots, R\}$.

Depending on the relationship between the number of available data-pairs N and the number R of model-points in the neural LUT, the following cases arise:

– **Case** $R < N$: The model constitutes a downsampling/decimation of the original dataset;
– **Case** $R = N$: The model constitutes a uniform[1] resampling of the original dataset;
– **Case** $R > N$: The model constitutes an interpolation of the original dataset.

By means of the dataset transform rule (8) and the isotonic statistical modeling procedure devised in [11],

[1] It is tacitly assumed that, while the points in the dataset $\mathbb{D}$ generally are not evenly spaced, the resampled coordinate $\hat{x}$ is evenly spaced in $\mathbb{M}$, although such an assumption is not strictly necessary for the discussed modeling algorithm to work.

it is possible to implement a statistical modeling procedure for non-monotonic relationships. The Algorithm 1 lists a pseudo-code implementation of the whole modeling procedure. Such pseudo-code is based on high-level functions summarized in the Table 1. A MATLAB-based code is attached as Appendix A.

It is worth underlining that switching from a monotonic dependency to a non-monotonic dependency causes the loss of some interesting features of the neural algorithm developed in [11]. In particular, we refer explicitly to the following facts:

- The relationship between *monotonic* datasets may be recovered even when there are incomplete data in the dataset as, for instance, incomplete records of the type $(\cdot, y_i)$. The existence of missing records in a dataset is not uncommon. This happens, for example, when readouts in a sensor network become temporarily unavailable due to communication loss or signal corruption [26]. Conversely, modeling a *non-monotonic* relationship requires complete datasets and no missing data may be handled in the context of *non-isotonic* modeling.
- The relationship between *monotonic* data may be recovered in the presence of a random reshuffling of the data-records. For example, a bivariate dataset $\{(x_i, y_i)|i = 1,\ 2,\ \ldots,\ N\}$ may come under the form of two unpaired datasets $\{x_i|i = 1,\ 2,\ \ldots,\ N\}$ and $\{y_j|j = 1,\ 2,\ \ldots,\ N\}$, where there is no known pairing between each sample x_i and y_j. This situation is also not uncommon in applications (see, for example, a problem arising in medical statistics described in [15]). A neural statistical *isotonic* modeling algorithm is able to infer the relationship underlying unpaired samples. As opposed to that, modeling a *non-monotonic* relationship requires a correct pairing of the data-records.

The proposed statistical modeling technique exhibits a number of distinguishing features. While the data pre-processing technique is data-driven, the underlying modeling method does not make direct use of the samples in the dataset as it is based on the extraction of collective information from the data; as a consequence, the model does not fit directly the data samples (which may be unreliable due to measurement errors or other hidden/nuisance variables) and results unfaithful in a mean-squared-error sense. The modeling procedure strives to capture the overall structure of the underlying phenomenon. Moreover, the proposed procedure does not make any assumption on the shape of the model, including monotonicity, that was the principal limitation of the previously-devised neural statistical modeling method [11]. As a result, the model may work on a wide range of datasets and there is no need to choose any functional dependency beforehand, in contrast to parametric/maximum-likelihood estimation methods. In addition, the involved probability density functions, the inferred model and the data-transform operator are represented by fully-learnable neural look-up tables. The probability density functions are estimated via occurrence histograms and the associated cumulative distribution functions are estimated by cumulative sums. The data-transform as well as the sought non-linear model are estimated by algebraic operations on such LUTs.

4 Numerical experiments

The present section illustrates some numerical features of the devised neural modeling algorithm.

A preliminary experiment aims at comparing the computational complexity of the proposed algorithm with the complexity exhibited by three neural learning rules for data monotonization recently proposed in [14] and summarized in the Subsection 2.2, in order to clearly render how the novel monotonization rule results advantageous in terms of execution time.

Further numerical experiments were chosen to illustrate the behavior of the devised statistical modeling procedure and were motivated by a contour resampling/interpolation problem that arises in shape analysis (for a review, see, e.g., [20]). The chosen datasets[2] are challenging and the experiments aim at illustrating objectively the merits and demerits of the proposed neural modeling method.

4.1 Preliminary experiment on empirical complexity evaluation

The equations describing the proposed data-monotonization learning rule look much simpler than the rules proposed in the previous contributions [14], not only because the new transformation rule is linear instead of being non-linear, but also because the new transformation only requires the calculation of a single parameter (the constant c). Allegedly, such combined advantages result in lighter computations and, hence, in shorter execution times.

In order to substantiate such argument, we compared the *Learning rule 1*, *Learning rule 2* and *Learning rule 3* from [14] with the *Three-stage learning rule*

[2] The shape-contour datasets used within the present paper were drawn from the *Surrey fish database* described in `http://www.ee.surrey.ac.uk/CVSSP/demos/css/demo.html`. Unfortunately, these datasets are no longer available from the server. We can make them available to interested readers upon request.

Algorithm 1 Three-stage neural modeling algorithm

function STATMOD(Sx, Sy, xx)
  $N \leftarrow$ **lenght**(Sx); ▷ Get cardinality of data sets
  $B \leftarrow$ **round**$(20 * log10(N))$; ▷ Compute number of bins
  for $i \leftarrow 2, N$ **do**
    $df[i] \leftarrow -(Sy[i] - Sy[i-1])/(Sx[i] - Sx[i-1])$;
  end for
  $c \leftarrow 1.01*$**max**(df); ▷ Compute the lifting constant
  for $j \leftarrow 1, N$ **do**
    $Sy[j] \leftarrow Sy[j] + c * Sx[j]$; ▷ Lift upward the y-data
  end for
  $[px, x] \leftarrow$**histogram**(Sx, B); ▷ Estimate the probability density functions
  $[py, y] \leftarrow$**histogram**(Sy, B);
  $DDx \leftarrow$ (**max**$(Sx)-$**min**$(Sx))/B$; ▷ Compuet bins' size
  $DDy \leftarrow$ (**max**$(Sy)-$**min**$(Sy))/B$;
  $px \leftarrow px/N$; ▷ Normalize the probability density functions
  $py \leftarrow (N * py + 1)/(B + N * N)$;
  $Px[1] \leftarrow 0$; ▷ Estimate the cumulative distribution functions
  $Py[1] \leftarrow 0$;
  $xsh[1] \leftarrow$ **min**(Sx);
  $ysh[1] \leftarrow$ **min**(Sy);
  for $k \leftarrow 2, N$ **do**
    $Px[k] \leftarrow px[k-1] + Px[k-1]$;
    $xsh[k] \leftarrow (x[k-1] + DDx)/2$;
    $Py[k] \leftarrow py[k-1] + Py[k-1]$;
    $ysh[k] \leftarrow (y[k-1] + DDy)/2$;
  end for
  $PPx \leftarrow$ **interpolate**(xsh, Px, xx); ▷ Estimate the model
  $yysh \leftarrow$ **interpolate**(Py, ysh, PPx);
  for $i \leftarrow 1, N$ **do** ▷ Shift downward the y-data
    $yy[i] \leftarrow yysh[i] - c * xx[i]$;
  end for
  return yy;
end function

Table 1 Functions used in the pseudo-coded Algorithm 1

Function name	Comment
max(x)	Allows to find the maximum element of a vector x.
min(x)	Allows to find the minimum element of a vector x.
[counts, centers] ← histogram(x, numbins)	Sorts vector x into n bins defined by numbins. The function returns two row vectors, counts containing the number of elements in each bin, and centers, indicating the location of each bin center on the x-axis.
interpolate(x, v, xq)	The function return the linear interpolated values from x, a vector of sample points, v contains the corresponding values, vector xq contains the coordinates of the query points.

described in Section 3. This comparison is based on computation times by a MATLAB© 2016a computing environment. The dataset chosen as a benchmark is the one referred to as *Data-set 2* in [14]. Such (synthetic) dataset was chosen because it affords the generation of as many data samples as desired. We ran the above four monotonization/demonotonization rules on a dataset counting an increasing number of samples, from 50 to 1000 with step 50, and recorded the execution time of each procedure. Each test was repeated 10 times on independent datasets and the recorded execution times were averaged to get rid of random fluctuations.

The result of such numerical analysis is displayed in the Figure 2. The top panel compares the execution times pertaining to the first (data monotonization) stage as required by the code to run. The bottom panel compares the execution times pertaining to the third (data de-monotonization) stage. Both stages (especially the third) of the novel method appear much quicker (i.e., computationally lighter) compared to the corresponding stages of the previously-proposed meth-

ods. In fact, the time-complexity of the last stage in the previous versions is around 10^{-4} seconds, while the time-complexity of the last stage in the current version is around 10^{-6} seconds, which is 2 orders of magnitude smaller.

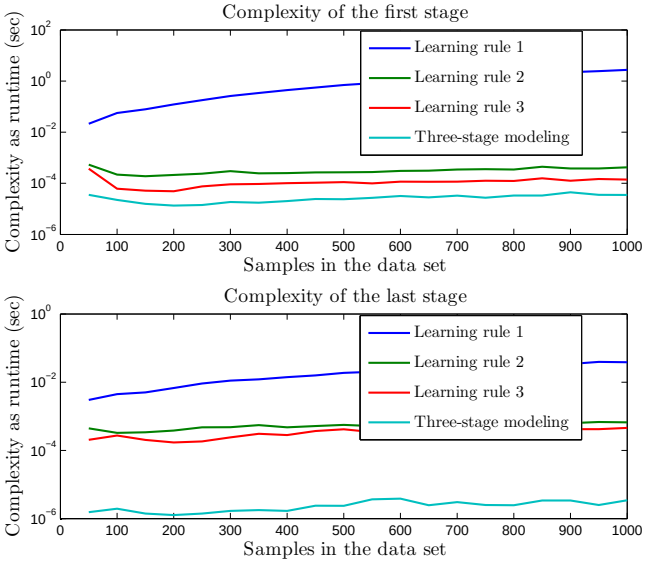

Fig. 2 Numerical comparison of the *Learning rule 1*, *Learning rule 2* and *Learning rule 3* from [14] with the *Three-stage learning rule*. Top panel: Monotonization stage. Bottom panel: De-monotonization stage.

The time-complexity comparison may be paired with a space-complexity evaluation. The equations describing the proposed data-monotonization learning rule look much simpler than the rules proposed in the previous contributions [14], because:

- The new transformation rule is *linear*, while the previous one is *non-linear*,
- The new transformation only requires *learning a single parameter* (the constant c) instead of *learning a whole neural activation function* (denoted by g in the summary presented in the Subsection 2.2).

The constant c is a scalar, whereas the function $g(x)$ needs to be evaluated at every point of the model. In terms of spatial complexity, the proposed monotonicity transform requires to store a single value in memory, while the previous methods [14] require to store an array of N elements, with N denoting again the cardinality of the dataset.

4.2 Experiments on a single contour resampling

The original dataset of interest consists of a set of pairs $\mathbb{S} = \{(\bar{\xi}_i, \bar{\eta}_i) \mid i = 1, \ldots, N\}$, each of which represents the coordinates of a point on a bi-dimensional contour (illustrated, for example, in the right-hand panel of Figure 3 in red color).

Since the points are not sampled uniformly along the contour, a coordinate $s_i \in [0,\ 1]$, hereafter termed *curvilinear coordinate*, was calculated on the basis of the available data. The curvilinear coordinate is a common coordinate system used to locate a point on a curved line based on the rectified distance from one of the end-points. (Note that we deal with closed curves, therefore, the endpoints coincide to one another.) The curvilinear coordinate for a shape-contour dataset $\mathbb{S}$ is calculated as illustrated in the pseudo-coded Algorithm 2. The pseudo-code defines a function that inputs

Algorithm 2 Algorithm to calculate the curvilinear coordinate associated to a shape contour

function CurvCoord(Sx, Sy)
  $s[1] \leftarrow 0$; ▷ Initialize curve's length
  for $i \leftarrow 2, N$ **do**
    $qx \leftarrow (Sx[i] - Sx[i-1]) * (Sx[i] - Sx[i-1])$;
    $qy \leftarrow (Sy[i] - Sy[i-1]) * (Sy[i] - Sy[i-1])$;
    $s[i] \leftarrow s[i-1] + \mathbf{sqrt}(qx + qy)$;
  end for ▷ Accumulate curve's length
  for $i \leftarrow 2, N$ **do**
    $s[i] \leftarrow s[i]/s[N]$;
  end for ▷ Normalize coordinate to $[0,\ 1]$
  return s;
end function

the dataset $\mathbb{S}$ in the form of two vectors (`Sx` for the $\bar{\xi}$ coordinates and `Sy` for the $\bar{\eta}$ coordinates) and returns a vector `s` for the curvilinear coordinates. The function `sqrt` returns the square root of its argument.

In this way, two separate datasets, namely, $\mathbb{S}_\xi = \{(s_i, \bar{\xi}_i) \mid i = 1, \ldots, N\}$ and $\mathbb{S}_\eta = \{(s_i, \bar{\eta}_i) \mid i = 1, \ldots, N\}$ were obtained. The neural statistical modeling procedure was applied separately to the dataset $\mathbb{S}_\xi$ to give the neural LUT model $\mathbb{M}_\xi = \{(\hat{s}_i, \hat{\xi}_i) \mid i = 1, \ldots, R\}$, and to the dataset $\mathbb{S}_\eta$ to give the neural LUT model $\mathbb{M}_\eta = \{(\hat{s}_i, \hat{\eta}_i) \mid i = 1, \ldots, R\}$, where R denotes the number of points chosen to model the data, as explained in Section 3. Note that, while the values s_i are generally unevenly distributed, the values $\hat{s}_i$ are uniformly spaced within the domain $[0,\ 1]$. In this dataset, the number of data-pairs is $N = 1036$. A model of the whole shape $\mathbb{S}$ is then obtained as $\mathbb{M} = \{(\hat{\xi}_i, \hat{\eta}_i) \mid i = 1, \ldots, R\}$.

It is to be noted how the datasets $(s_i, \bar{\xi}_i)$ and $(s_i, \bar{\eta}_i)$, chosen to test the discussed neural modeling procedure, are quite complex, therefore the estimation of the marginal probability density functions of the involved variables is challenging. For the considered dataset, in fact, the estimated occurrence histograms for the vari-

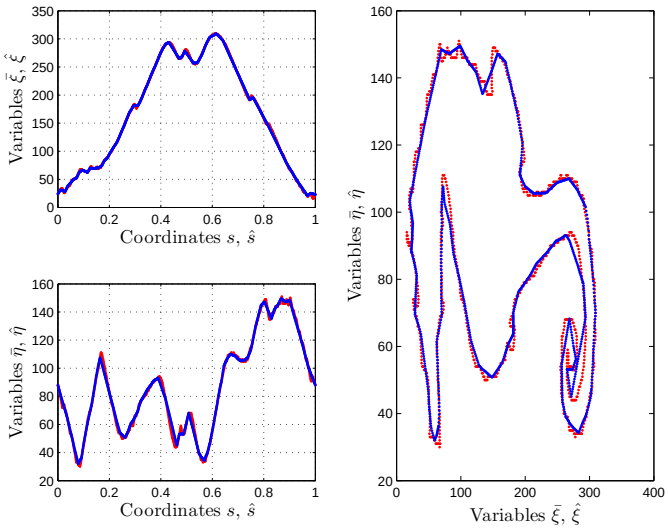

Fig. 3 Illustration of the bidimensional contour dataset ('Sea Horse'), referred to as *Shape 1* dataset. Results of Experiment 1 – Left-top panel: Original dataset $(s_i, \bar{\xi}_i)$ (red dots) and neural LUT model $(\hat{s}_i, \hat{\xi}_i)$ (blue line). Left-bottom panel: Original dataset $(s_i, \bar{\eta}_i)$ (red dots) and neural LUT model $(\hat{s}_i, \hat{\eta}_i)$ (blue line). Right-hand panel: Original contour $\mathbb{S}$ (red dots) and inferred model $\mathbb{M}$ (blue line).

ables $\bar{\xi}$ and $\bar{\eta}$, after being transfomed by the rule (10), are shown in the Figure 4.

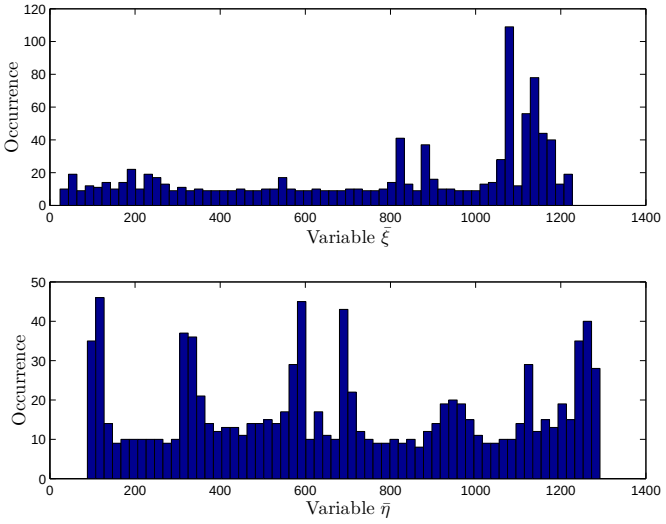

Fig. 4 *Shape 1* dataset: Estimated occurrence histograms for the variables $\bar{\xi}$ and $\bar{\eta}$ after being transformed by the method (10). The number of bins calculated by the procedure of Algorithm 1 is $B = 61$.

In order to quantify the features of the proposed statistical modeling method, several figures-of-demerit were defined. In particular, we used: *Root mean squared error*, *Mean absolute error*, *Root mean squared roughness*, and *Coefficient of determination*, as measures of discrepancy between a dataset $\mathbb{S}$ and the neural model $\mathbb{M}$. Note that a coordinate pair in the neural LUT model $(\hat{\xi}_i, \hat{\eta}_i)$ may not be compared directly with the data-pair $(\bar{\xi}_i, \bar{\eta}_i)$, in general, because of the different underlying resampling of the curvilinear coordinates s_i and $\hat{s}_i$. Therefore, a model-pair $(\hat{\xi}_i, \hat{\eta}_i) \in \mathbb{M}$ is compared with the data-pair $(\xi_i^c, \eta_i^c) \in \mathbb{S}$ that corresponds to the *closest value of the curvilinear coordinate*. For each value of the index $j = 1, 2, \ldots, R$, a pair (ξ_j^c, η_j^c) is assigned the pair $(\hat{\xi}_i, \hat{\eta}_i)$ (namely $\xi_j^c \leftarrow \hat{\xi}_i$ and $\eta_j^c \leftarrow \hat{\eta}_i$) whose index $i = 1, 2, \ldots, N$ is the index of the curvilinear coordinate s_i which is the nearest to the curvilinear coordinate value $\hat{s}_j$ of the model-pair $(\hat{\xi}_j, \hat{\eta}_j)$. The number of points (ξ_j^c, η_j^c) coincides, therefore, to the number R of model-points. A pseudo-code is displayed in the Algorithm 3. The function described in the pseudo-code inputs a shape-contour, represented by three arrays `Sx`, `Sy` and `s` (which denote the ξ-data, the η-data and the associated curvilinear coordinate, respectively) and a LUT-based neural model of such shape-contour, represented by three arrays `hSx`, `hSy` and `hs` (which denote the ξ, η-model and the associated curvilinear coordinate, respectively). This function returns two arrays `cSx` and `cSy`, which represent a set of points over the given shape-contour closest to the points provided by the model in terms of curvilinear coordinates. Note that it is assumed that the entries in both arrays `s` and `hs` are sorted in ascending order.

The mentioned figures-of-demerit are defined in the Figure 5:

- The 'root mean squared error' measures the average discrepancy between an inferred model and the original dataset in terms of squared differences.
- The 'root mean squared roughness' (or 'root mean square gradient') measures the average roughness of a model in terms of magnitude of its first-order derivative. We devised this novel measure on the basis of a figure-of-quality used in surface analysis (see, for example, [9])[3].
- The 'mean absolute error' measures the average discrepancy between an inferred model and the original dataset in a way similar to the RMSE, except that it is based on absolute differences rather than squared differences.
- The coefficients of determination R_ξ^2 and R_η^2, are numbers used in the context of statistical modeling,

[3] The idea behind the RMSR index is as follows. Let us denote by $z(x)$ the profile ordinate of a line parametererized by x. A completely flat (i.e. minimally rough) profile will be characterized by $dz/dx = 0$, hence the cumulative quantity $\int (dz/dx)^2 dx = 0$. Conversely, the more a profile is uneven/rough, the larger $(dz/dx)^2$ is, the larger $\int (dz/dx)^2 dx$ will result. We took a numerical approximation of this integral as a measure of roughness of a shape, approximated numerically by the sum of terms $(\hat{\xi}_i - \hat{\xi}_{i-1})^2$ and $(\hat{\eta}_i - \hat{\eta}_{i-1})^2$.

Algorithm 3 Algorithm to calculate the closest points in a model corresponding to a given dataset

function CLOSEST(Sx, Sy, s, hSx, hSy, hs)
  $N \leftarrow$ **length**(s); ▷ Get dataset cardinality
  $R \leftarrow$ **length**(hs); ▷ Get model size
  $cSx[1] \leftarrow Sx[1]$;
  $cSy[1] \leftarrow Sy[1]$; ▷ First point coincides
  $i \leftarrow 2$;
  for $j \leftarrow 2, N$ **do**
    while $(s[i] < hs[j])$ & $(i < N)$ **do**
      $i \leftarrow i + 1$;
    end while ▷ Scan the vector hs until the smallest value less than $s[i]$
    if $s[i] - hs[j] < hs[j] - s[i-1]$ **then**
      $cSx[j] \leftarrow Sx[i]$;
      $cSy[j] \leftarrow Sy[i]$;
    else
      $cSx[j] \leftarrow Sx[i-1]$;
      $cSy[j] \leftarrow Sy[i-1]$;
    end if ▷ Choose the closest value ($s[i]$ or $s[i-1]$)
  end for
  return cSx, cSy;
end function

$$\text{Root mean squared error (RMSE)} = \sqrt{\frac{1}{R}\sum_{i=1}^{R}((\hat{\xi}_i - \xi_i^{\mathrm{c}})^2 + (\hat{\eta}_i - \eta_i^{\mathrm{c}})^2)},$$

$$\text{Root mean squared roughness (RMSR)} = \sqrt{\frac{1}{R-1}\sum_{i=2}^{R}((\hat{\xi}_i - \hat{\xi}_{i-1})^2 + (\hat{\eta}_i - \hat{\eta}_{i-1})^2)},$$

$$\text{Mean absolute error (MAE)} = \frac{1}{R}\sum_{i=1}^{R}(|\hat{\xi}_i - \xi_i^{\mathrm{c}}| + |\hat{\eta}_i - \eta_i^{\mathrm{c}}|),$$

$$\text{Coefficient of determination } R_\xi^2 = 1 - \frac{\sum_{i=1}^{R}(\hat{\xi}_i - \xi_i^{\mathrm{c}})^2}{\sum_{i=1}^{R}(\xi_i^{\mathrm{c}} - \langle\xi^{\mathrm{c}}\rangle)^2},\ \langle\xi^{\mathrm{c}}\rangle = \frac{1}{R}\sum_{i=1}^{R}\xi_i^{\mathrm{c}},$$

$$\text{Coefficient of determination } R_\eta^2 = 1 - \frac{\sum_{i=1}^{R}(\hat{\eta}_i - \eta_i^{\mathrm{c}})^2}{\sum_{i=1}^{R}(\eta_i^{\mathrm{c}} - \langle\eta^{\mathrm{c}}\rangle)^2},\ \langle\eta^{\mathrm{c}}\rangle = \frac{1}{R}\sum_{i=1}^{R}\eta_i^{\mathrm{c}}.$$

Fig. 5 Figures-of-demerit of a regression algorithm.

whose main purpose is hypothesis testing. It provides a measure of how well observed outcomes are replicated by the model, based on the proportion of total variation of outcomes explained by the model [16].

Experiment 1. In the first experiment, an even resampling is illustrated, namely, $R = N$, as shown in the Figure 3. The statistical modeling procedure was compared with *linear interpolation* and with *cubic-spline interpolation*[4]. The obtained results are displayed in the Table 2. The statistical model exhibits the largest RMSE and MAE figures compared to linear interpolation and spline interpolation. On the other hand, the statistical model exhibits the lowest RMSR figure compared with linear interpolation and spline interpolation (which returns a curly model due to the underlying cubic spline). The values of the coefficients of determination corresponding to the statistical model are slightly smaller than 1. While the data-modeling procedure based on *linear or spline interpolation follows the data closely*, the neural LUT model returned by *the statistical inference algorithm tries to capture the in-*

Table 2 Experiment on even resampling on *Shape 1* dataset: Results of modeling by statistical regression compared with linear and spline interpolation

Indexes	Linear	Spline	StatMod
RMSE	0.3438	0.3450	2.8996
RMSR	1.1194	1.1428	0.9642
MAE	0.3576	0.3637	2.8326
R_ξ^2	1.0000	1.0000	0.9996
R_η^2	1.0000	1.0000	0.9958

[4] We used the MATLAB function **interp1** with the syntax `yh = interp1(x,y,xh,method)`, where `(x,y)` is a dataset, `yh` is the result of interpolation at 'query points' `xh`, and `method` is either '`linear`' or '`spline`'.

formation content buried in the data and produces a smoother model, which is deemed a desirable feature.

Experiment 2. In the second experiment, a downsampling is illustrated, namely, $R < N$. First, it was chosen $R = 100$. The Figure 6 renders the result of such downsampling. The statistical modeling procedure

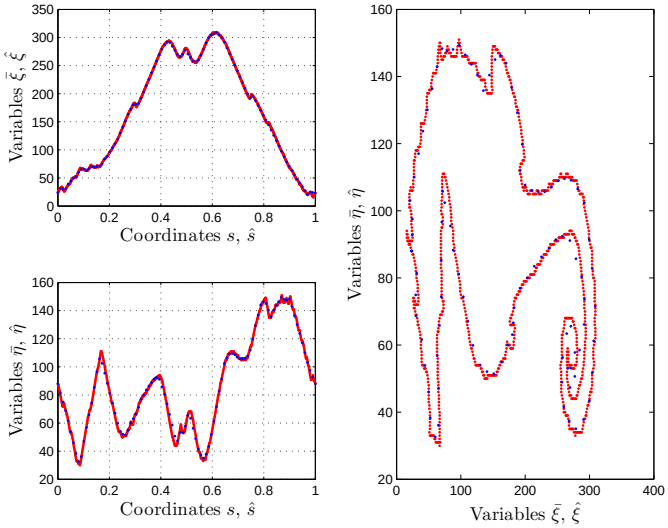

Fig. 6 *Shape 1* dataset: Results of the Experiment 2 – Case $R = 100$. Left-top panel: Original dataset $(s_i, \bar{\xi}_i)$ (red dots) and neural LUT model $(\hat{s}_i, \hat{\xi}_i)$ (blue line). Left-bottom panel: Original dataset $(s_i, \bar{\eta}_i)$ (red dots) and neural LUT model $(\hat{s}_i, \hat{\eta}_i)$ (blue line). Right-hand panel: Original contour $\mathbb{S}$ (red dots) and inferred model $\mathbb{M}$ (blue line).

was compared with linear interpolation and with cubic-spline interpolation. The obtained results are displayed in the Table 3. Second, it was chosen $R = 500$. The Figure 7 shows the result of such downsampling. A comparison of the statistical modeling procedure with linear and with cubic-spline interpolation is displayed in the Table 4. In both cases, the RMSE and the MAE pertaining to the statistical modeling method is much higher than the RMSE pertaining to the linear and to the spline interpolation, while the RMSR is lower. The values of the coefficients of determination appear comparable. It can be concluded that the statistical

Table 3 Experiment on downsampling ($R = 100$) the *Shape 1* dataset: Results of modeling by statistical regression compared with linear and spline interpolation

Indexes	Linear	Spline	StatMod
RMSE	0.3868	0.3933	2.8814
RMSR	10.4275	10.4383	9.8066
MAE	0.4192	0.4309	2.8278
R^2_ξ	1.0000	1.0000	0.9996
R^2_η	0.9999	0.9999	0.9957

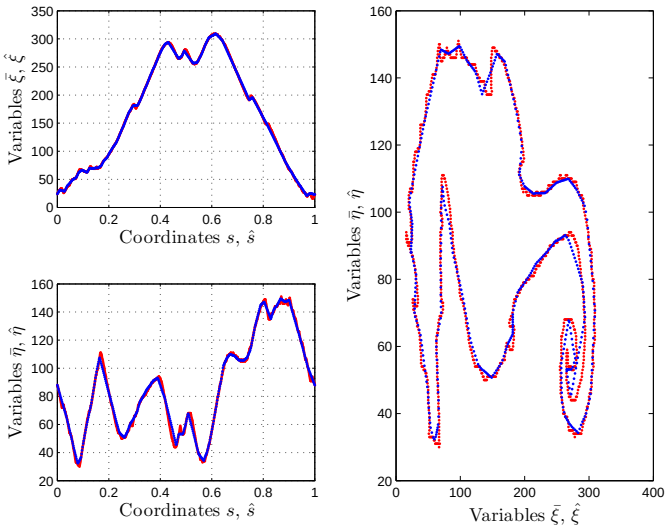

Fig. 7 *Shape 1* dataset: Results of the Experiment 2 – Case $R = 500$. Left-top panel: Original dataset $(s_i, \bar{\xi}_i)$ (red dots) and neural LUT model $(\hat{s}_i, \hat{\xi}_i)$ (blue line). Left-bottom panel: Original dataset $(s_i, \bar{\eta}_i)$ (red dots) and neural LUT model $(\hat{s}_i, \hat{\eta}_i)$ (blue line). Right-hand panel: Original contour $\mathbb{S}$ (red dots) and inferred model $\mathbb{M}$ (blue line).

Table 4 Experiment on downsampling ($R = 500$) the *Shape 1* dataset: Results of modeling by statistical regression compared with linear and spline interpolation

Indexes	Linear	Spline	StatMod
RMSE	0.3492	0.3511	2.9048
RMSR	2.2599	2.2817	1.9925
MAE	0.3619	0.3705	2.8588
R^2_ξ	1.0000	1.0000	0.9996
R^2_η	0.9999	0.9999	0.9957

modeling returns an unfaithful model, with respect to the original data, while keeping the computed model smoother than the original data.

An advantage of getting a smoother neural LUT model is that it enables computing more accurately two functions associated with the evenly downsampled contour, namely, the angular function $\theta(\hat{s})$, whose samples are computed by means of the approximation

$$\theta_i = \tan^{-1}\left(\frac{\hat{\eta}_i - \hat{\eta}_{i-1}}{\hat{\xi}_i - \hat{\xi}_{i-1}}\right), \tag{12}$$

and its derivative with respect to the curvilinear coordinate $\hat{s}$, $\theta'(\hat{s})$, whose samples are computed numerically by

$$\theta'_i = \frac{\theta_i - \theta_{i-1}}{\Delta\hat{s}}, \tag{13}$$

with $\Delta\hat{s} = \frac{1}{R-1}$ denoting the spacing between two adjacent points (in terms of their curvilinear coordinates). The angular function as well as its first-order deriva-

tive[5] are instrumental in the analysis connected to the geometrization of the shape space [20]. The results of the estimation of the angular function and of its derivative on the basis of the models obtained by spline interpolation, linear interpolation and statistical modeling with $R = 500$ are shown in the Figure 8. The angular

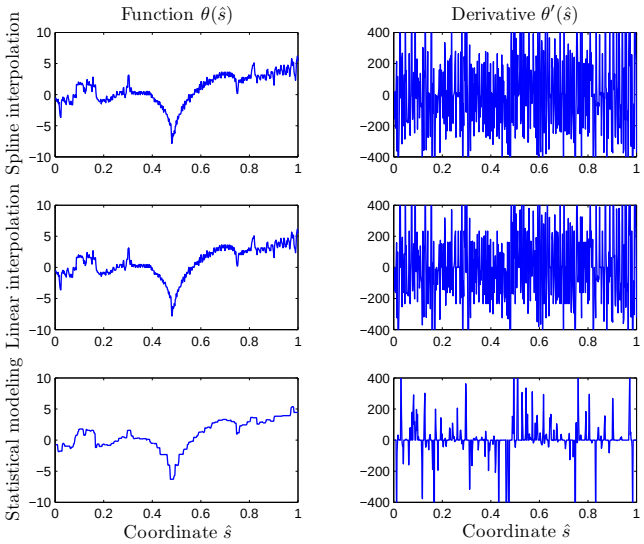

Fig. 8 *Shape 1* dataset: Results of the Experiment 2 – Case $R = 500$. Left-hand columns: Angular function $\theta(\hat{s})$. Right-hand column: Derivative $\theta'(\hat{s})$. First row: Results obtained with cubic spline interpolation. Second row: Results obtained by linear interpolation. Third row: Results obtained by statistical modeling.

function θ estimated by the statistical model is much smoother than the estimates obtained by means of a linear/spline downsampling, which, in turn, enables a more accurate estimation of the derivative θ'.

Experiment 3. In the third experiment, the sampling rate was set to $R > N$. The first trial concerns the case that $R = \lfloor \frac{10}{4} N \rfloor$, the second considered trial concerns the case that $R = \lfloor \frac{10}{7} N \rfloor$ and the third trial refers to the situation in which $R = \lfloor \frac{10}{9} N \rfloor$. The results for such three cases are summarized in the Table 5. The obtained results suggest that, as for the previous experiments, the proposed neural system learns a smooth model.

4.3 Comparative experiments on four shape-contours

While the previous three experiments were conducted on the *Shape 1* dataset only, in this last numerical experiment we compared the behavior of the considered

[5] The derivative $\kappa(s) = \theta'(s)$ represents the local curvature of the planar shape described by the orientation function $\theta(s)$.

Table 5 Experiment on interpolation on the *Shape 1* dataset: Results of modeling by statistical regression compared with linear and spline interpolation

Indexes (Resampling)	**Linear**	**Spline**	**StatMod**
RMSE ($R = \lfloor (10/4)N \rfloor$)	0.5517	0.5530	4.5724
RMSR ($R = \lfloor (10/4)N \rfloor$)	0.7198	0.7363	0.6105
MAE ($R = \lfloor (10/4)N \rfloor$)	0.9097	0.9246	7.1294
R^2_ξ ($R = \lfloor (10/4)N \rfloor$)	1.0000	1.0000	0.9996
R^2_η ($R = \lfloor (10/4)N \rfloor$)	1.0000	1.0000	0.9958
RMSE ($R = \lfloor (10/7)N \rfloor$)	0.4153	0.4154	3.4601
RMSR ($R = \lfloor (10/7)N \rfloor$)	0.9438	0.9660	0.8067
MAE ($R = \lfloor (10/7)N \rfloor$)	0.5188	0.5261	4.0808
R^2_ξ ($R = \lfloor (10/7)N \rfloor$)	1.0000	1.0000	0.9996
R^2_η ($R = \lfloor (10/7)N \rfloor$)	1.0000	1.0000	0.9958
RMSE ($R = \lfloor (10/9)N \rfloor$)	0.3632	0.3639	3.0396
RMSR ($R = \lfloor (10/9)N \rfloor$)	1.0647	1.0878	0.9143
MAE ($R = \lfloor (10/9)N \rfloor$)	0.3975	0.4038	3.1593
R^2_ξ ($R = \lfloor (10/9)N \rfloor$)	1.0000	1.0000	0.9996
R^2_η ($R = \lfloor (10/9)N \rfloor$)	1.0000	1.0000	0.9958

statistical modeling algorithms on four further shape-contour datasets, illustrated in the Figure 9. Moreover, in the first test below, we compare the modeling capabilities of the proposed *Three-stage neural modeling algorithm* with those exhibited by the *Learning rule 1*, *Learning rule 2* and *Learning rule 3* proposed in the previous publication [14]. (It is, perhaps, useful to underline that a comparison with the earlier method proposed in [11] is not feasible, since the shape-contour datasets are markedly non-monotonic, while the statistical modeling algorithm proposed in [11] is only able to cope with monotonic datasets.)

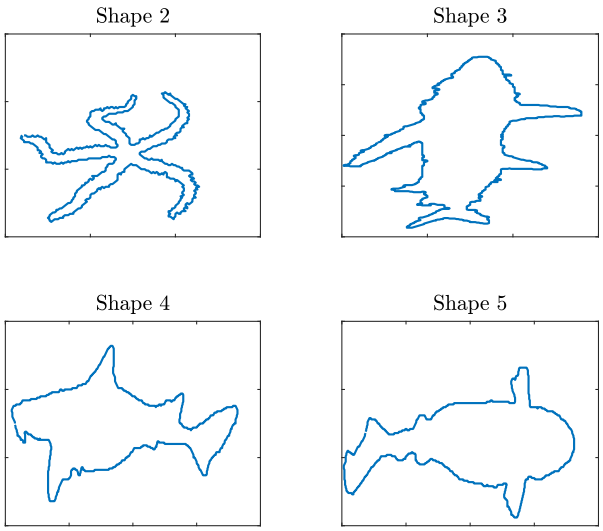

Fig. 9 Illustration of the *Shape 2*, *Shape 3*, *Shape 4* and *Shape 5* datasets.

Experiment 1. In the first experiment, again an even resampling is illustrated, namely, we set $R = N$.

The statistical modeling procedure was compared with linear interpolation and with cubic-spline interpolation. Since the three neural modeling algorithms introduced in the previous work [14] (and summarized in the Subsection 2.2) can cope with the case $R = N$, we also compared the statistical modeling procedure with these three methods.

The obtained results are displayed in the Table 6. It is interesting to see how similar shapes get similar values of the performance indexes as, for example, *Shape 4* and *Shape 5*. Conversely, contour-shapes that appear quite different from one another as, for example, *Shape 2* and *Shape 3*, get quite different values of the error and roughness indexes.

Experiment 2. In the second experiment, an even downsampling is illustrated, namely, $R < N$. First, it was chosen $R = 100$. The statistical modeling procedure was compared with linear interpolation and with cubic-spline interpolation. The obtained results are displayed in the Table 7. Second, it was chosen $R = 500$. A comparison of the statistical modeling procedure with linear and with cubic-spline interpolation is displayed in the Table 8.

Experiment 3. In the third experiment, the sampling rate was set again to $R > N$ with three different ratios R/N. The results for such three cases are summarized in the Table 9. The results suggest that, as for the previous experiments, the proposed neural-LUT algorithm learns a smooth model.

5 Conclusion

The present paper introduces a novel neural statistical modeling procedure. The discussed modeling method is based on a previously-introduced isotonic modeling technique that is able to capture only monotonic dependencies and whose scope was extended to non-monotonic data through a linear data-transformation technique.

The resulting procedure is computationally light and fast to execute. Experimental results show that the estimated model is smoother than the models obtained with linear and cubic-spline-based interpolation techniques (as well as the models obtain by means of the previous neural modeling technique proposed by the first author and colleagues in [14]). This feature makes the proposed technique be suitable to the estimation of model derivatives. In summary, the main contributions of the present paper with respect to the previous works [11,14] may be outlined as follows:

- The present neural modeling algorithm is able to deal with bivariate non-monotonic relationships, while the previous contribution [11] could only deal with monotonic relationships;
- The proposed neural modeling algorithm allows for an arbitrary resampling rate R/N, while the previous contribution [14] could only deal with the case $R = N$;
- The proposed modeling technique performs data-monotonization by a simple linear transformation of the data, while the previous contribution [14] involved a complex multiplicative transform to be learned from the dataset. As a result, the spatial complexity as well as the time complexity of the current modeling method are considerably lower.

A current limitation of the discussed technique is that it may be applied to bivariate data only. Future endeavors will be devoted to the extension of the trivariate isotonic modeling method, which works for three variables [12,13], to the non-monotonic trivariate case. In addition, a research endeavor toward the extension of the discussed techniques to non-stationary (i.e., time-varying) datasets is currently being pursued.

Acknowledgements

The authors wish to gratefully thank the anonymous reviewers whose thorough reading of the manuscript and detailed comments helped greatly in improving the quality of this paper.

Conflict of interest

The authors declare that they have no conflict of interest.

References

1. Abdolahzare Z, Mehdizadeh SA (2016). Nonlinear mathematical modeling of seed spacing uniformity of a pneumatic planter using genetic programming and image processing. Neural Computing and Applications. Accepted for publication. Available online at the publisher's website since 12 July 2016 (doi:10.1007/s00521-016-2450-1)
2. Barlow RE, Brunk HD (1972) The isotonic regression problem and its dual. Journal of the American Statistical Association 67(337): 140 – 147
3. Conolly RB, Lutz WK (2003) Nonmonotonic dose-response relationships: Mechanistic basis, kinetic modeling, and implications for risk assessment. Toxicological Sciences 77: 151 – 157
4. Domínguez-Menchero JS, González-Rodríguez G (2007) Analyzing an extension of the isotonic regression problem. Metrika 66: 19 – 30
5. Duren RW, Marks II RJ, Reynolds PD, Trumbo ML (2007) Real-time neural network inversion on the SRC-6e reconfigurable computer. IEEE Transactions on Neural Networks 18(3): 889 – 901

Table 6 Comparative experiment on the even resampling on four datasets (*Shape 2*, *Shape 3*, *Shape 4* and *Shape 5*): Results of modeling by statistical regression ('StatMod') compared with linear interpolation ('Linear'), spline interpolation ('Spline'), modeling by means of Learning rule 1 ('LR1'), Learning rule 2 ('LR2') and Learning rule 3 ('LR3').

Dataset	Indexes	Linear	Spline	StatMod	LR 1	LR 2	LR 3
Shape 2	**RMSE**	0.3611	0.3625	2.1248	3.2274	0.4105	2.7211
	RMSR	1.1400	1.2002	0.9340	2.1116	1.1413	1.1722
	MAE	0.3996	0.3945	2.2295	1.5785	0.4558	1.1834
	R^2_ξ	0.9999	0.9999	0.9972	0.9938	0.9999	0.9909
	R^2_η	0.9999	0.9999	0.9964	0.9914	0.9998	0.9998
Shape 3	**RMSE**	0.3622	0.3638	2.7554	7.5293	0.3792	0.9086
	RMSR	1.1037	1.1313	0.9483	2.6707	1.1049	1.1299
	MAE	0.3767	0.3853	2.6727	2.0080	0.4075	0.7231
	R^2_ξ	0.9999	0.9999	0.9971	0.9988	0.9999	0.9995
	R^2_η	1.0000	1.0000	0.9995	0.9937	1.0000	1.0000
Shape 4	**RMSE**	0.3553	0.3553	2.8871	5.3497	0.3788	11.0277
	RMSR	1.0971	1.1164	0.9940	1.1220	1.0982	1.1993
	MAE	0.3638	0.3718	2.6705	3.6699	0.4044	4.4877
	R^2_ξ	1.0000	1.0000	0.9996	0.9982	1.0000	0.9889
	R^2_η	0.9999	0.9999	0.9948	0.9862	0.9999	0.9999
Shape 5	**RMSE**	0.3452	0.3446	2.3774	3.6605	0.3885	1.2550
	RMSR	1.0944	1.1132	1.0109	1.0950	1.0955	1.1028
	MAE	0.3501	0.3569	2.3857	1.8266	0.4137	0.9424
	R^2_ξ	1.0000	1.0000	0.9997	0.9993	1.0000	0.9999
	R^2_η	0.9999	0.9999	0.9967	0.9925	0.9999	0.9998

Table 7 Experiment on downsampling ($R = 100$) four datasets (*Shape 2*, *Shape 3*, *Shape 4* and *Shape 5*): Results of modeling by statistical regression ('StatMod') compared with linear interpolation ('Linear') and spline interpolation ('Spline').

Dataset	Indexes	Linear	Spline	StatMod
Shape 2	**RMSE**	0.3507	0.3601	2.1571
	RMSR	6.3548	6.3762	5.7882
	MAE	0.3894	0.3986	2.2668
	R^2_ξ	0.9999	0.9999	0.9973
	R^2_η	0.9999	0.9999	0.9961
Shape 3	**RMSE**	0.3283	0.3326	2.7852
	RMSR	10.0707	10.0781	9.3886
	MAE	0.3241	0.3343	2.6766
	R^2_ξ	1.0000	0.9999	0.9969
	R^2_η	1.0000	1.0000	0.9995
Shape 4	**RMSE**	0.3551	0.3621	2.8376
	RMSR	8.8802	8.8909	8.4300
	MAE	0.3610	0.3688	2.6140
	R^2_ξ	1.0000	1.0000	0.9996
	R^2_η	0.9999	0.9999	0.9949
Shape 5	**RMSE**	0.3474	0.3495	2.4103
	RMSR	8.8151	8.8229	8.4344
	MAE	0.3668	0.3730	2.4377
	R^2_ξ	1.0000	1.0000	0.9997
	R^2_η	0.9999	0.9999	0.9966

Table 8 Experiment on downsampling ($R = 500$) four datasets (*Shape 2*, *Shape 3*, *Shape 4* and *Shape 5*): Results of modeling by statistical regression ('StatMod') compared with linear interpolation ('Linear') and spline interpolation ('Spline').

Dataset	Indexes	Linear	Spline	StatMod
Shape 2	**RMSE**	0.3631	0.3618	2.1091
	RMSR	1.4386	1.5031	1.1942
	MAE	0.3979	0.3900	2.2167
	R^2_ξ	0.9999	0.9999	0.9973
	R^2_η	0.9999	0.9999	0.9964
Shape 3	**RMSE**	0.3387	0.3419	2.7357
	RMSR	2.1908	2.2155	1.9310
	MAE	0.3461	0.3557	2.6516
	R^2_ξ	1.0000	1.0000	0.9972
	R^2_η	1.0000	1.0000	0.9995
Shape 4	**RMSE**	0.3383	0.3389	2.8916
	RMSR	1.8560	1.8771	1.7005
	MAE	0.3412	0.3475	2.6614
	R^2_ξ	1.0000	1.0000	0.9996
	R^2_η	0.9999	0.9999	0.9948
Shape 5	**RMSE**	0.3336	0.3330	2.3504
	RMSR	1.8281	1.8443	1.7094
	MAE	0.3329	0.3397	2.3676
	R^2_ξ	1.0000	1.0000	0.9997
	R^2_η	0.9999	0.9999	0.9968

6. El-Shafie A, Abdelazim T, Noureldin A (2010) Neural network modeling of time-dependent creep deformations in masonry structures. Neural Computing and Applications 19(4): 583 – 594
7. Esaki L, Arakawa Y, Kitamura M (March 2010) Esaki diode is still a radio star, half a century on. Nature 464: 31
8. Evrendilek F (2014) Assessing neural networks with wavelet denoising and regression models in predicting diel dynamics of eddy covariance-measured latent and sensible heat fluxes and evapotranspiration. Neural Computing and Applications 24(2): 327 – 337
9. Feng Y-J, Lin Z-X, Zhang R-Z (2011) The influence of root mean square phase gradient of continuous phase plate on smoothing focal spot Acta Physica Sinica 60(10): 104202

Table 9 Experiment on interpolation on the four datasets (*Shape 2*, *Shape 3*, *Shape 4* and *Shape 5*): Results of modeling by statistical regression ('StatMod') compared with linear interpolation ('Linear') and spline interpolation ('Spline').

Dataset	Indexes	Linear	Spline	StatMod
Shape 2	**RMSE** ($R = \lfloor(10/4)N\rfloor$)	0.5821	0.5878	3.3560
	RMSR ($R = \lfloor(10/4)N\rfloor$)	0.7538	0.7945	0.5928
	MAE ($R = \lfloor(10/4)N\rfloor$)	1.0156	1.0071	5.6132
	R^2_ξ ($R = \lfloor(10/4)N\rfloor$)	0.9999	0.9999	0.9972
	R^2_η ($R = \lfloor(10/4)N\rfloor$)	0.9999	0.9999	0.9964
	RMSE ($R = \lfloor(10/7)N\rfloor$)	0.4391	0.4415	2.5380
	RMSR ($R = \lfloor(10/7)N\rfloor$)	0.9776	1.0317	0.7826
	MAE ($R = \lfloor(10/7)N\rfloor$)	0.5751	0.5691	3.2041
	R^2_ξ ($R = \lfloor(10/7)N\rfloor$)	0.9999	0.9999	0.9972
	R^2_η ($R = \lfloor(10/7)N\rfloor$)	0.9999	0.9999	0.9964
	RMSE ($R = \lfloor(10/9)N\rfloor$)	0.3776	0.3824	2.2344
	RMSR ($R = \lfloor(10/9)N\rfloor$)	1.0934	1.1517	0.8860
	MAE ($R = \lfloor(10/9)N\rfloor$)	0.4356	0.4312	2.4824
	R^2_ξ ($R = \lfloor(10/9)N\rfloor$)	0.9999	0.9999	0.9973
	R^2_η ($R = \lfloor(10/9)N\rfloor$)	0.9999	0.9999	0.9963
Shape 3	**RMSE** ($R = \lfloor(10/4)N\rfloor$)	0.5434	0.5459	4.3403
	RMSR ($R = \lfloor(10/4)N\rfloor$)	0.7117	0.7296	0.6007
	MAE ($R = \lfloor(10/4)N\rfloor$)	0.8864	0.9050	6.6366
	R^2_ξ ($R = \lfloor(10/4)N\rfloor$)	1.0000	1.0000	0.9972
	R^2_η ($R = \lfloor(10/4)N\rfloor$)	1.0000	1.0000	0.9995
	RMSE ($R = \lfloor(10/7)N\rfloor$)	0.4115	0.4124	3.2788
	RMSR ($R = \lfloor(10/7)N\rfloor$)	0.9336	0.9574	0.7939
	MAE ($R = \lfloor(10/7)N\rfloor$)	0.5033	0.5133	3.7962
	R^2_ξ ($R = \lfloor(10/7)N\rfloor$)	1.0000	1.0000	0.9972
	R^2_η ($R = \lfloor(10/7)N\rfloor$)	1.0000	1.0000	0.9995
	RMSE ($R = \lfloor(10/9)N\rfloor$)	0.3641	0.3681	2.8950
	RMSR ($R = \lfloor(10/9)N\rfloor$)	1.0522	1.0779	0.8994
	MAE ($R = \lfloor(10/9)N\rfloor$)	0.3956	0.4058	2.9565
	R^2_ξ ($R = \lfloor(10/9)N\rfloor$)	1.0000	1.0000	0.9971
	R^2_η ($R = \lfloor(10/9)N\rfloor$)	1.0000	1.0000	0.9995
Shape 4	**RMSE** ($R = \lfloor(10/4)N\rfloor$)	0.5363	0.5374	4.5676
	RMSR ($R = \lfloor(10/4)N\rfloor$)	0.7047	0.7177	0.6297
	MAE ($R = \lfloor(10/4)N\rfloor$)	0.8620	0.8798	6.6372
	R^2_ξ ($R = \lfloor(10/4)N\rfloor$)	1.0000	1.0000	0.9996
	R^2_η ($R = \lfloor(10/4)N\rfloor$)	0.9999	0.9999	0.9948
	RMSE ($R = \lfloor(10/7)N\rfloor$)	0.4063	0.4072	3.4501
	RMSR ($R = \lfloor(10/7)N\rfloor$)	0.9262	0.9434	0.8327
	MAE ($R = \lfloor(10/7)N\rfloor$)	0.4898	0.5002	3.7842
	R^2_ξ ($R = \lfloor(10/7)N\rfloor$)	1.0000	1.0000	0.9996
	R^2_η ($R = \lfloor(10/7)N\rfloor$)	0.9999	0.9999	0.9948
	RMSE ($R = \lfloor(10/9)N\rfloor$)	0.3560	0.3571	3.0406
	RMSR ($R = \lfloor(10/9)N\rfloor$)	1.0450	1.0631	0.9438
	MAE ($R = \lfloor(10/9)N\rfloor$)	0.3812	0.3889	2.9260
	R^2_ξ ($R = \lfloor(10/9)N\rfloor$)	1.0000	1.0000	0.9996
	R^2_η ($R = \lfloor(10/9)N\rfloor$)	0.9999	0.9999	0.9947
Shape 5	**RMSE** ($R = \lfloor(10/4)N\rfloor$)	0.5334	0.5334	3.7175
	RMSR ($R = \lfloor(10/4)N\rfloor$)	0.7028	0.7154	0.6403
	MAE ($R = \lfloor(10/4)N\rfloor$)	0.8549	0.8716	5.9462
	R^2_ξ ($R = \lfloor(10/4)N\rfloor$)	1.0000	1.0000	0.9997
	R^2_η ($R = \lfloor(10/4)N\rfloor$)	0.9999	0.9999	0.9968
	RMSE ($R = \lfloor(10/7)N\rfloor$)	0.4045	0.4036	2.8144
	RMSR ($R = \lfloor(10/7)N\rfloor$)	0.9234	0.9403	0.8464
	MAE ($R = \lfloor(10/7)N\rfloor$)	0.4878	0.4964	3.4005
	R^2_ξ ($R = \lfloor(10/7)N\rfloor$)	1.0000	1.0000	0.9997
	R^2_η ($R = \lfloor(10/7)N\rfloor$)	0.9999	0.9999	0.9968
	RMSE ($R = \lfloor(10/9)N\rfloor$)	0.3368	0.3384	2.4813
	RMSR ($R = \lfloor(10/9)N\rfloor$)	1.0423	1.0601	0.9589
	MAE ($R = \lfloor(10/9)N\rfloor$)	0.3517	0.3601	2.6479
	R^2_ξ ($R = \lfloor(10/9)N\rfloor$)	1.0000	1.0000	0.9997
	R^2_η ($R = \lfloor(10/9)N\rfloor$)	0.9999	0.9999	0.9968

(in Chinese)
10. Fiori S (2002) Hybrid independent component analysis by adaptive LUT activation function neurons. Neural Networks 15(1): 85 – 94
11. Fiori S (2013) Fast statistical regression in presence of a dominant independent variable. Neural Computing and Applications 22: 1367 – 1378
12. Fiori S (2013) An isotonic trivariate statistical regression method. Advances in Data Analysis and Classification 7: 209 – 235
13. Fiori S (2014) Fast closed-form trivariate statistical isotonic modeling. Electronics Letters 50: 708 – 710
14. Fiori S, Gong T, Lee HK (December 2015) Bivariate non-isotonic statistical regression by a look-up table neural system. Cognitive Computation 7(6): 715 – 730
15. Goetghebeur E, Lapp K (1997) The effect of treatment compliance in a placebo-controlled trial: Regression with unpaired data. Journal of the Royal Statistical Society - Series C (Applied Statistics) 46(3): 351 – 364
16. Gujarati DN, Porter DC (2009) Basic Econometrics, Fifth edition, New York: McGraw-Hill/Irwin. 73 – 78 (ISBN: 978-0-07-337577-9)
17. Guo WW, Xue H (2012) An incorporative statistic and neural approach for crop yield modelling and forecasting. Neural Computing and Applications 21(1): 109 – 117
18. Hájek P, Olej V (2011) Credit rating modelling by kernel-based approaches with supervised and semi-supervised learning. Neural Computing and Applications 20(6): 761 – 773
19. Isermann R, Münchhof M (2011) Neural networks and Lookup Tables for identification. In Identification of Dynamic Systems – An Introduction with Applications. 501 – 537, Springer Berlin Heidelberg (ISBN: 978-3-540-78878-2)
20. Klassen E, Srivastava A, Mio W (2004) Analysis of planar shapes using geodesic paths on shape spaces. IEEE Transactions on Pattern Analysis and Machine Intelligence 26: 372 – 383
21. Kotłowski W, Słowiński R (2009) Rule learning with monotonicity constraints. In Proceedings of the 26$^{\text{th}}$ International Conference on Machine Learning (Montreal, Canada) 537 – 544
22. Laudani A, Lozito GM, Riganti Fulginei F, Salvini A (2015) On training efficiency and computational costs of a feed forward neural network: A review. Computational Intelligence and Neuroscience 2015: Article ID 818243, 13 pages (doi:10.1155/2015/818243)
23. Luss R, Rosset S (2014) Generalized isotonic regression. Journal of Computational and Graphical Statistics 23(1): 192 – 210
24. Motulsky HJ, Ransnas LA (1987) Fitting curves to data using nonlinear regression: a practical and nonmathematical review. Federation of American Societies for Experimental Biology (FASEB) Journal 1(5): 365 – 374
25. Najah A, El-Shafie A, Karim OA, El-Shafie AH (2013) Application of artificial neural networks for water quality prediction. Neural Computing and Applications 22(1): 187 – 201
26. Niu K, Zhao F, Qiao X (2013) A missing data imputation algorithm in wireless sensor network based on minimized similarity distortion. In Proc. of the Sixth International Symposium on Computational Intelligence and Design (Hangzhou, People's Republic of China, October 28-29, 2013) 2: 235 – 238
27. Owolabi TO, Zakariya YF, Olatunji SO, Akande KO (2016) Estimation of melting points of fatty acids using homogeneously hybridized support vector regression. Neural Computing and Applications. Accepted for publication. Available online at the publisher's website since May 19, 2016 (doi:10.1007/s00521-016-2344-2)
28. Prabhu S, Uma M, Vinayagam BK (2015) Surface roughness prediction using Taguchi-fuzzy logic-neural network analysis for CNT nanofluids based grinding process. Neural Computing and Applications 26(1); 41 – 55

29. Saâdaoui F (2017) A seasonal feedforward neural network to forecast electricity prices. Neural Computing and Applications 28(4), 835 – 847
30. Simsion G, Witt G (2005) Data Modeling Essentials. 3rd Edition, Morgan Kauffman Publishers
31. Smith M (1996) Neural Networks for Statistical Modeling. International Thomson Computer Press
32. Specht DE (1991) A general regression neural network. IEEE Transactions on Neural Networks 2(6): 568 – 576
33. Tamminen S, Laurinen P, Röning J (1999) Comparing regression trees with neural networks in aerobic fitness approximation. In Proceedings of the International Computing Sciences Conference Symposium on Advances in Intelligent Data Analysis (Rochester (NY, USA), 1999) 414 – 419
34. Tibshirani RJ, Hoefling H, Tibshirani R (2011) Nearly-isotonic regression. Technometrics 53(1): 54 – 61
35. Vansteenkiste E (2015) Lookup-table based neurons for an efficient FPGA implementation of neural networks. Master Thesis of the Computing Systems Lab (CSL), Electronics and Information Systems (ELIS) department, Ghent University (Belgium)
36. Wu J, Meyer MC, Opsomer JD (2015) Penalized isotonic regression. Journal of Statistical Planning and Inference 161: 1 – 24
37. Yu R, Leung PS, Bienfang P (2006) Predicting shrimp growth: Artificial neural network versus nonlinear regression models. Aquacultural Engineering 34(1): 26 – 32

A MATLAB Implementation of the novel regression procedure

A MATLAB© implementation of the modeling procedure devised in the present research and illustrated in the Algorithm 3 is reported in the Figure 10. The MATLAB© code illustrates the simple structure of the proposed method.

The MATLAB© function inputs the triple (`Sx,Sy,xx`) as three arrays. The array-pair (`Sx,Sy`) represents the dataset $\mathbb{D}$ to be modeled as a look-up table, the array `xx` represents the set of $\hat{x}$-values whose corresponding $\hat{y}$-values are sought[6]. The function returns an array `yy` that represents the estimated model $\mathbb{M}$ as a LUT neural network (`xx,yy`).

In the above version of the statistical isotonic modeling procedure, the number of subdivisions for probability density function estimation is automatically selected by the rules in the command line 5 and does not coincide necessarily with the number R of partitions of the x-axes for model estimation. The command line 7 computes the lifting-upward constant c in equation (8), while the command lines 9 and 26 perform the lifting upward/downward, respectively. The lines 11–24 are borrowed from the isotonic modeling procedure described in [11].

[6] The above procedure takes the same syntax of the MATLAB©'s function 'interp1'.

```
1  function yy = statmod(Sx,Sy,xx)
2  % Cardinality of data sets
3  N=length(Sx);
4  % Numbers of bins
5  B=ceil(20*log10(N));
6  % Lifting constant
7  c=1.01*max(-diff(Sy)./diff(Sx));
8  % Lifting upward the y-data
9  Sy=Sy+c*Sx;
10 % Estimation of the pdf's
11 [px,x]=hist(Sx,B);
12 [py,y]=hist(Sy,B);
13 % Compute bins' size
14 DDx=(max(Sx)-min(Sx))/B;
15 DDy=(max(Sy)-min(Sy))/B;
16 % Pdf's normalization
17 px=px/N;
18 py=(N*py+1)/(B+N^2);
19 % Estimation of the cdf's
20 Px=[0 cumsum(px)]; xsh=[min(Sx) x+DDx/2];
21 Py=[0 cumsum(py)]; ysh=[min(Sy) y+DDy/2];
22 % Estimation of the model
23 PPx=interp1(xsh,Px,xx,'linear','extrap');
24 yysh=interp1(Py,ysh,PPx,'linear','extrap');
25 % Shifting downward the y-data
26 yy=yysh-c*xx;
```

Fig. 10 MATLAB© code that implements the three-stage neural modeling algorithm.