KYBERNETIKA — VOLUME 43 (2007), NUMBER 4, PAGES 547 – 559

PIECEWISE APPROXIMATION AND NEURAL NETWORKS

Martina Révayová and Csaba Török

The paper deals with the recently proposed autotracking piecewise cubic approximation (APCA) based on the discrete projective transformation, and neural networks (NN). The suggested new approach facilitates the analysis of data with complex dependence and relatively small errors. We introduce a new representation of polynomials that can provide different local approximation models. We demonstrate how APCA can be applied to especially noisy data thanks to NN and local estimations. On the other hand, the new approximation method also has its impact on neural networks. We show how APCA helps to decrease the computation time of feed forward NN.

Keywords: data smoothing, least squares and related methods, linear regression, approximation by polynomials, neural networks

AMS Subject Classification: 93E14, 93E24, 62J05, 41A10, 62M45

1. INTRODUCTION

Methods of approximation and the associated problems are of wide interest in data analysis and information gain process. To address these problems diverse approaches have been proposed. On one side there are the parametric methods, such as regression and splines [3, 7], and on the other side the nonparametric ones, such as wavelets and neural networks (NN) that do not result in functional equations [4, 6].

The nature of dependence and accuracy of data points define the tools and techniques the researchers can use. Recently a new approach to the analysis of complex dependence with relatively small noise has been proposed [2]. The suggested autotracking piecewise cubic approximation divides the interval/curve into subintervals/segments of various lengths (stage 1) and gives a technique for obtaining integral cubic approximants (stage 2). Finding the breakpoints in an autotracking mode and the iterative computation scheme of approximants are the two main features of the proposed method that uses a special approximation model.

It is well known that if the measurements of a polynomial $f(x)$ of degree p are taken with equidistant step, then the difference $\Delta f_k = f_{k+1} - f_k$ yields a polynomial of degree $p-1$, and the difference of second order $\Delta^2 f_k$ is a polynomial of degree $p-2$. We will see that the discrete projective transformation (DPT), based on which APCA is developed, decreases the polynomial degree by two, regardless the step's character.

The paper deals with APCA and NN. Our aim is to show that when one needs to describe the dependence of strongly noised data with equations, APCA can be used due to NN, too. But not only NN can help in applying APCA. We show how the application of APCA results in speeding up the computation of certain types of NN.

Section two provides the definitions of the forward and the backward DPT and some enhanced properties of the transformation. Its main result is the representation theorem. Section three gives a brief introduction to APCA and the underlying theory. The next section is devoted to neural networks. Part 5.1 shows how to approximate noisy data based on NN and APCA. Here we use a new modified version of APCA. The last part of the paper describes how to decrease the computation time of NN.

The algorithms were implemented in Visual $C\#$ based on the component library LinAlg described in [10].

2. DISCRETE PROJECTIVE TRANSFORMATION

The discrete projective transformation is a new operation, forward and backward, over a continuous function defined by a formula or a table. The DPT was introduced by N.D.Dikoussar [1]. The DP transformation of a function $f(x)$ is defined by two pivot points $[x_1, f(x_1)]$, $[x_2, f(x_2)]$ and the abscissa of a fix point $[x_0, f(x_0)]$. Although DPT is a four point transformation (the fourth one is $[x, f(x)]$), the equations and formulas of methods based on DPT use only three of them, $[x_0, f(x_0)]$ is absent [1, 2, 8, 9]. It is possible due to two further transformations. Before applying the specified methods, the data points are shifted along the x axis the way that the origin of the new coordinate system will be at x_0. At the end of the process the resulting data are transformed back to the original coordinates. There are two main arguments for such a process. The formulae with three points are simpler and computation with data scattered along zero may be in many cases preferable.

This section is devoted to a short description of DPT. The definitions are reformulated and the enhanced results are derived using $[x_0, f(x_0)]$. The main benefit of using formulas with $[x_0, f(x_0)]$ is a straight proof of the representation theorem.

Definition 1. The forward DPT of any continuous function $f(x)$ based on arbitrary two pivot points $[x_1, f(x_1)]$, $[x_2, f(x_2)]$ and x_0 is given by

$$Df(x) = h_0(x)f(x) + h_1(x)f(x_1) + h_2(x)f(x_2), \tag{1}$$

where $x_0 \neq x_1 \neq x_2$, $x \neq x_1 \neq x_2$ and

$$h_0(x) = \frac{(x_0 - x_1)(x_0 - x_2)}{(x - x_1)(x - x_2)}, \quad h_1(x) = \frac{(x_0 - x)(x_0 - x_2)}{(x_1 - x)(x_1 - x_2)}, \tag{2}$$
$$h_2(x) = \frac{(x_0 - x)(x_0 - x_1)}{(x_2 - x)(x_2 - x_1)}.$$

Remark 1. As we can see the Definition 1 does not use $f(x_0)$. The point $[x_0, f(x_0)]$ is a special one, its ordinate appears in the result of the transformation, see Lemma 1 and Theorem 1.

Definition 2. The backward DPT of any continuous function $f(x)$ based on arbitrary two pivot points $[x_1, f(x_1)]$, $[x_2, f(x_2)]$ and x_0 is defined by

$$D^{-1}Df(x) = p_0(x)Df(x) + p_1(x)f(x_1) + p_2(x)f(x_2), \tag{3}$$

where $x_0 \neq x_1 \neq x_2$ and

$$p_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}, \quad p_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}, \tag{4}$$
$$p_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}.$$

From (2) and (4) it follows that

$$p_0(x) = \frac{1}{h_0(x)}, \quad p_1(x) = -\frac{h_1(x)}{h_0(x)}, \quad p_2(x) = -\frac{h_2(x)}{h_0(x)},$$

and

$$h_0(x) + h_1(x) + h_2(x) = p_0(x) + p_1(x) + p_2(x) = 1.$$

Hence and from the Definition 1 we get one of the main features of DPT, the linearity

$$\begin{aligned} D(af(x)+b) &= h_0(x)(af(x)+b) + h_1(x)(af(x_1)+b) + h_2(x)(af(x_2)+b) \\ &= aDf(x) + b(h_0(x) + h_1(x) + h_2(x)) = aDf(x) + b. \end{aligned}$$

The properties of DP transformation with $x_0 = 0$ are studied most detailed for power functions and polynomials [8, 9]. Although we are dealing with cubic approximation we give general formulas for power functions and polynomials of degree $p \geq 3$.

From Definition 1 after some algebraic computations we get

Lemma 1. Let p be a nonnegative integer. The DP transformation of the power function x^p is

a) a constant for $p = 0, 1, 2$

$$Dx^p = x_0^p,$$

b) a polynomial of degree $p-2$ for $p \geq 3$

$$Dx^p = x_0^p + z_1(x) \sum_{i=0}^{p-3} R_i x^{p-3-i}, \tag{5}$$

where

$$z_1(x) = (x-x_0)(x_1-x_0)(x_2-x_0) \tag{6}$$

and

$$R_i = \sum_{k_0=0}^{i} \sum_{k_1=0}^{i-k_0} x_0^{k_0} x_1^{k_1} x_2^{i-k_0-k_1}, \; i \in Z_0^+.$$

Proof. b) Let $q_i = \sum_{j=0}^{i} x_1^j x_2^{i-j} = R_i - x_0 R_{i-1}$. From the Definition 1 we obtain

$$Dx^p = h_0(x)x^p + h_1(x)x_1^p + h_2(x)x_2^p.$$

Since

$$h_0(x)x^p = \frac{(x_0-x_1)(x_0-x_2)}{(x-x_1)(x-x_2)}x^p,$$

$$\frac{x^p}{(x-x_1)(x-x_2)} = \sum_{i=0}^{p-2} q_i x^{p-2-i} + \frac{x_1^p}{(x-x_1)(x_1-x_2)} - \frac{x_2^p}{(x-x_2)(x_1-x_2)},$$

and

$$\begin{aligned} &\frac{(x_0-x_1)(x_0-x_2)}{(x-x_1)(x_1-x_2)}x_1^p - \frac{(x_0-x_1)(x_0-x_2)}{(x-x_2)(x_1-x_2)}x_2^p + h_1(x)x_1^p + h_2(x)x_2^p \\ = \;& \frac{x_0-x_2}{x_1-x_2}x_1^p - \frac{x_0-x_1}{x_1-x_2}x_2^p, \end{aligned}$$

we get

$$\begin{aligned} Dx^p &= (x_0-x_1)(x_0-x_2)\sum_{i=0}^{p-2} q_i x^{p-2-i} \\ &\quad + \frac{x_0-x_2}{x_1-x_2}x_1^p - \frac{x_0-x_1}{x_1-x_2}x_2^p + x_0^p - x_0^p \\ &= x_0^p + (x_0-x_1)(x_0-x_2)\left(\sum_{i=0}^{p-2} q_i x^{p-2-i} + R_{p-2}\right). \end{aligned}$$

For the completion of the proof it is sufficient to mention that

$$(x-x_0)\sum_{i=0}^{p-3} R_i x^{p-3-i} = \sum_{i=0}^{p-2} q_i x^{p-2-i} + R_{p-2}. \qquad \square$$

Notice the different use of x_0^p in a) and b). In (5) the use of x_0^p enables the factorization, however it is canceled after simplification.

Let us denote the sum in (5) by

$$T_i(x) = \sum_{j=0}^{i} R_j x^{i-j},\ i \in Z_0^+. \tag{7}$$

The following two lemmas show that both R_i and T_i can be computed either by recursion or as a dot product.

Lemma 2. R_i, $i \in Z_0^+$, can be computed

a) recursively

$$R_i = r_i + x_2 R_{i-1},\ R_0 = 1,$$

b) as a dot product

$$R_i = (x_0, x_1, x_2) \circ (x_0^{i-1}, r_{i-1}, R_{i-1}),$$

or

$$R_i = (1, x_2, \ldots, x_2^{i-1}, x_2^i) \circ (r_i, r_{i-1}, \ldots, r_1, 1),$$

where $r_i = x_0^i + x_1 r_{i-1},\ r_0 = 1,$

or $r_i = (1, x_0, \ldots, x_0^{i-1}, x_0^i) \circ (x_1^i, x_1^{i-1}, \ldots, x_1, 1).$

Lemma 3. $T_i(x)$, $i \in Z_0^+$, can be computed

a) recursively

$$T_i(x) = xT_{i-1}(x) + R_i,\ T_0(x) = 1,$$

b) as a dot product

$$T_i(x) = (1, x, \ldots, x^{i-1}, x^i) \circ (R_i, R_{i-1}, \ldots, R_1, 1).$$

Example 1. Let us see some examples of r_i, R_i and the DP transformation of power functions:

$$\begin{aligned}
r_1 &= x_0 + x_1 r_0 = x_0 + x_1,\\
r_2 &= x_0^2 + x_1 r_1 = x_0^2 + x_1(x_0 + x_1),\\
R_1 &= r_1 + x_2 R_0 = x_0 + x_1 + x_2,\\
R_2 &= r_2 + x_2 R_1 = x_0^2 + x_1(x_0 + x_1) + x_2(x_0 + x_1 + x_2),\\
Dx^3 &= x_0^3 + z_1(x)\underbrace{R_0}_{T_0(x)},\\
Dx^4 &= x_0^4 + z_1(x)\underbrace{(R_0 x + R_1)}_{T_1(x)},\\
Dx^5 &= x_0^5 + z_1(x)\underbrace{(R_0 x^2 + R_1 x + R_2)}_{T_2(x))}.
\end{aligned}$$

Lemma 1 and the linearity of D imply that D decreases the degree of a polynomial

$$P_p(x) = \sum_{i=0}^{p} a_i x^i$$

by two. The exact formula is given by

Theorem 1. Let p be a nonnegative integer. The DP transformation of a polynomial $P_p(x)$ is

a) a constant for $p = 0, 1, 2$

$$DP_p(x) = P_p(x_0),$$

b) a polynomial of degree $p - 2$ for $p \geq 3$

$$DP_p(x) = P_p(x_0) + z_1(x) \sum_{i=3}^{p} a_i T_{i-3}(x). \tag{8}$$

Proof.

b) From Lemma 1 and linearity of D we get

$$\begin{aligned}
DP_p(x) &= \sum_{i=0}^{p} a_i D(x^i) = \sum_{i=0}^{2} a_i D(x^i) + \sum_{i=3}^{p} a_i D(x^i)\\
&= \sum_{i=0}^{2} a_i x_0^i + \sum_{i=3}^{p} a_i (x_0^i + z_1(x) T_{i-3}(x)) = P_p(x_0) + z_1(x) \sum_{i=3}^{p} a_i T_{i-3}(x).
\end{aligned}$$

□

Remark 2. The exact formulae for the DPT of power functions and polynomials were given by Dikoussar and Török in [1] and [8], respectively. Their result is gained under the assumption $x_0 = 0$. The result in [8] differs from (8) in two additional points: it uses matrices and a slightly different iterative schema for the computation of $T_i(x)$.

Example 2. From (8) we get

$$\begin{aligned} DP_3(x) &= P_3(x_0) + z_1(x)a_3, \\ DP_4(x) &= P_4(x_0) + z_1(x)(a_3 + a_4T_1(x)), \\ DP_5(x) &= P_5(x_0) + z_1(x)(a_3 + a_4T_1(x) + a_5T_2(x)). \end{aligned}$$

Based on the backward DPT and the previous theorem we get a new representation of polynomials $P = I + ZA$

Theorem 2. Let p be an integer greater than or equals three. Then the polynomial $P_p(x)$ can be expressed in the form

$$P_p(x) = I(x) + Z(x)A(x), \tag{9}$$

where

$$\begin{aligned} I(x) &= p_0(x)f(x_0) + p_1(x)f(x_1) + p_2(x)f(x_2), \\ Z(x) &= (x - x_0)(x - x_1)(x - x_2), \\ A(x) &= \sum_{i=3}^{p} a_i T_{i-3}(x). \end{aligned} \tag{10}$$

Before proving this result let us notice that I is a classical interpolating polynomial, A is a polynomial of degree $p - 3$, and

$$I(x_0) = f(x_0), I(x_1) = f(x_1), I(x_2) = f(x_2),$$

$$Z(x_0) = 0, Z(x_1) = 0 \text{ and } Z(x_2) = 0.$$

Proof. Based on the backward DP transformation D^{-1}, Theorem 1 and the assumption about P_p we get:

$$\begin{aligned} P_p(x) &= D^{-1}(DP_p(x)) = p_0(x)DP_p(x) + p_1(x)P_p(x_1) + p_2(x)P_p(x_2) \\ &= p_0(x)\left(P_p(x_0) + z_1(x)\sum_{i=3}^{p} a_i T_{i-3}(x)\right) + p_1(x)f(x_1) + p_2(x)f(x_2) \\ &= I(x) + p_0(x)z_1(x)A(x) = I(x) + Z(x)A(x). \end{aligned}$$

□

3. APCA

The autotracking piecewise cubic approximation is based upon a cubic approximation model with a free parameter α

$$f \approx I + Z\alpha, \tag{11}$$

proposed in [1, 2]. The right hands side of (11) is a particular case of the representation theorem. It is get from (9) for $p = 3$. Although one can leverage instead of the single approximation parameter α more general approximants A given by (10), this paper uses for modeling in place of the representation (9) of polynomials of degree greater than three the simplest one (11).

The model (11) and its components are illustrated in Figure 1, where the polynomial $f(x) = -36 + 96x - 97x^2 + 47x^3 - 11x^4 - x^5$ is approximated by the cubic approximant $C = I + Z\alpha$ over the interval $[x_1, x_2]$, $x_1 = 1.1$ and $x_2 = 1.8$. In addition to $f(x)$ and $C(x)$, the functions $I(x)$ and $Z(x)\alpha$ are also plotted ($\alpha = 4.38833$) and you can see their behavior in the arguments x_0, x_1 and x_2. Although the role of x_0, x_1, x_2 can be played in the DPT by any three real different values, in APCA x_1 and x_2 are associated with the left and right end of the given subinterval, respectively, and x_0 belongs to $[x_1, x_2]$.

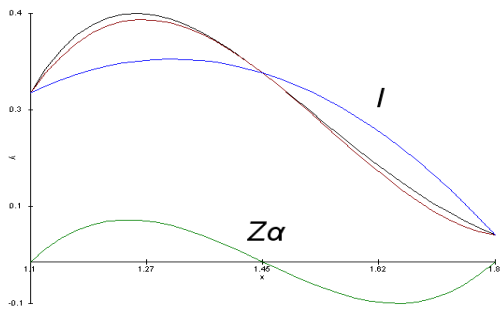

Fig. 1. Approximation by model $f \approx I(x) + Z(x)\alpha$.

The following codelines in MS Visual $C\#$ illustrate a fragment of the algorithm of computation and plotting of the APCA approximant – notice that in addition to the data X, Y one has also to provide information about the three pivot points:

```
xy = FileRead("XY.txt");
n  = xy.RowsCount;                    // the count of points
apca = new IZA(xy, 0, (n-1)/2, n-1);  // the indices of pivots
apca.YApproximant.Plot();
```

APCA has two stages. The first stage results in the subintervals and the second one in the cubic approximants. The model

$$C(x) = I(x) + Z(x)\alpha$$

with a free parameter α, parabola $I(x)$ and cubic parabola $Z(x)$, is used in both stages of APCA. The main task in stage 1 is to detect subintervals, over which the function/data can be approximated by cubic polynomials. The estimation of α is computed by a recursive least-squares method based on the model (11) and the criterion for knot detection uses a control parameter δ, see [2] and also Section 5.

Once the right end of the interval is detected, an integral cubic approximant can be constructed in various ways (stage 2). To ensure the continuity of two neighbour approximants, splines can be used or once again the model (11). The nice feature of the model is that it comprises both interpolation (I) and approximation ($Z\alpha$). $I(x)$ is computed by the two endpoints of the interval and an arbitrary inner point, and either the corresponding function values/measurements or their approximations. To complete the construction of the integral estimation $C(x)$ the unknown α can be estimated for example by regression based on $Y - I = Z\alpha$ or by a method proposed in [2].

4. FEED–FORWARD NEURAL NETWORKS

In the theory of NN the two most used methods for approximation and prediction are feed forward neural networks (FFNN) and radial basis functions. We shortly describe the former. Figure 2 depicts FFNN with n input neurons, one hidden layer with h neurons, and m output neurons.

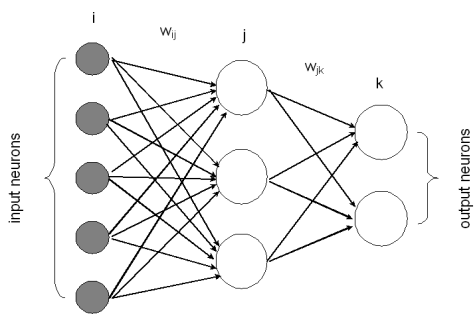

Fig. 2. Feed-forward neural network, $n = 5,\ h = 3,\ m = 2$.

The weighted input of the jth hidden neuron is computed as

$$u_j = t_j + \sum_{i=1}^{n} w_{ij} x_i,\ \ j = \overline{1, h},$$

where t_j is the threshold value of the hidden neuron, w_{ij} are the weights of the connections between neurons from neighbour layers, and x_i is the ith input. A hidden neuron transforms its weighted input by activation function. Nonlinear activation functions for hidden layers may be e. g.:

- $\varphi(x) = \dfrac{1}{1+e^{-x}}$ standard sigmoid,
- $\varphi(x) = \dfrac{1-e^{-x}}{1+e^{-x}}$ hyperbolic tangent type.

The output of the jth hidden neuron can be written as

$$y_j = \varphi(u_j) = \varphi\left(t_j + \sum_{i=1}^{n} w_{ij}x_i\right).$$

The activation functions for output neurons are mostly linear functions. The NN from Figure 2 can be expressed by

$$y_k = t_k + \sum_{j=1}^{h} w_{jk}y_j = t_k + \sum_{j=1}^{h} w_{jk}\varphi\left(t_j + \sum_{i=1}^{n} w_{ij}x_i\right),\ k = \overline{1,m}.$$

Consider a set of training patterns $T = \{[\boldsymbol{x}^l, \boldsymbol{y}^l], l = 1, \ldots, p\}$. Our aim is to teach the NN based on the input $\boldsymbol{x}^l = (x_1, \ldots, x_n)$ and output $\boldsymbol{y}^l = (y_1, \ldots, y_m)$ vectors from the training set, i. e. evaluate the estimations of weights and thresholds $\boldsymbol{w}$, $\boldsymbol{t}$. The error function of NN is defined by

$$E(\boldsymbol{w}, \boldsymbol{t}) = \frac{1}{2}\sum_{l=1}^{p}\left\|\boldsymbol{y}^l - \hat{\boldsymbol{y}}^l\right\|^2,$$

where $\hat{\boldsymbol{y}}^l$ is the estimated output of NN for input $\boldsymbol{x}^l$. The weights and thresholds are estimated by minimization of the error function. To detect the minimum of function $E(\boldsymbol{w}, \boldsymbol{t})$ one can use the steepest descent method. The change in weights can be expressed as

$$w_{ij} = w_{ij} - lr\frac{\partial E}{\partial w_{ij}},$$

where lr is the learning rate.

In NN, approximation or prediction is made based on the gained estimations of $\boldsymbol{w}$, $\boldsymbol{t}$ and a new input vector $\boldsymbol{x}$. How approximation based on NN and APCA works is shown in the next part. We shall return to the activation functions in the last part.

5. APCA AND NEURAL NETWORKS

5.1. Noisy data

The present section suggests a new approach to smoothing data by autotracked piecewise splines based on the combination of NN, APCA and splines that result in local estimates and smoothing, reducing the number of segments and cubic polynomials, respectively.

Consider real data $D = \{[x_i, y_i + \epsilon_i] : i = \overline{1,211}\}$, where x gives the slip and y the resistance ratio $P/P_{\exp}$, with non equidistant step that are strongly destroyed with

errors, see Figure 3. Our aim is to express the functional dependence between x and y. Regression polynomials of order $7-9$ are either wavy or decline too rapidly at the right end, see Figure 3. Instead of searching the functional dependence based on the nonlinear regression let us look at the autotracking piecewise cubic approximation. Since the pivot points of APCA may be sensitive to irregularities, before applying APCA it is preferable either to use point estimations of the pivot points or smooth the data. We will follow the second case.

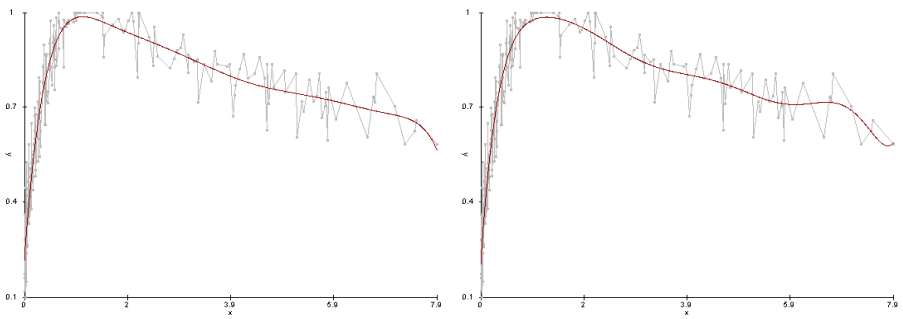

Fig. 3. Polynomials of degree $8, 9$.

There are several nonparametric smoothing techniques, such as wavelets, neural networks or Friedmans's SuperSmoother. For wavelet analysis there are few data points, and since NN and the supersmoother are similar techniques, we decided for NN.

The proper choice of the training data is a key moment in the learning process of NN. Let us denote the training set by

$$T = \{[\boldsymbol{x}^l, \boldsymbol{y}^l], l = \overline{1, p}\},$$

where p is the count of the training patterns, $\boldsymbol{x}^l = (x_{t_{l1}}, \ldots, x_{t_{lN}})$ is the input to the NN, $\boldsymbol{y}^l = (y_{t_{l1}}, \ldots, y_{t_{lN}})$ is the output and t_{lj} is the coordinate index of the original data, $l = \overline{1, p}$ and $j = \overline{1, N}$. Notice that the lth training pattern can be simply expressed by the vector of indices $T_l = (t_{l1}, \ldots, t_{lN})$, where $N < n$ is the length of the training pattern (the number of input and output neurons). We selected the indices of the training patterns (i. e. the training patterns themselves) based on sets of indices I_j, $j = \overline{1, N}$

- with the same number of indices
- with different number of indices.

The investigation showed that training patterns with intervals of equal lengths along the axis x (the second case) and $p = 20$ provided the most satisfactory smoothing.

To get a function dependence we applied the polynomial regression once again, however without acceptable result – the approximants remained wavy (we remind

that NN does not result in functional equation – it is hidden in the topology of NN and its parameters $\boldsymbol{w}$ and $\boldsymbol{t}$). APCA reduced the number of segments to five and the cubic splines constructed over them between the four verticals are shown in Figure 4. The data points on the left are more dense, hence they are zoomed-in in the right picture.

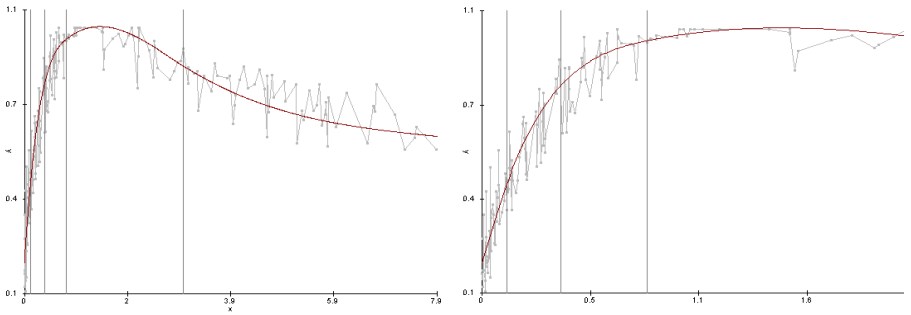

Fig. 4. NN-APCA-Spline approximant.

The overall process of executing the NN, APCA and construction of the spline approximants illustrates the following fragment of codelines

```
nnArgs.intervalsNb = 21;    nnArgs.trainSampNb = 20;
nnArgs.TD = Training.SamePointsNb;
nnArgs.delta = 0.008;
trDat  = new Training(X, Y, nnArgs);

nnAppr = trDat.NNApprox();
apca   = new Apca(nnAppr.X, nnAppr.Y, delta);
spline = Spline(X2, apca.SegmentEndX, apca.SegmentEndY);
spline.Plot();
```

5.2. Error free data

This section illustrates how the number of segments in APCA can be decreased thanks to a new way of selecting the position of the inner pivot point $[b, f(b)]$ and how the computation time of NN can be reduced using APCA approximants for the activation function.

The learning process of NN requires in some cases, such as pattern recognition or prediction based on many thousand data points, heavy computation [5]. In these cases every enhancement or modification that leads to the decrease of the computation time is valuable.

Feed forward neural networks by default use the computationally expensive activation function $\varphi(x) = \frac{1-e^{-x}}{1+e^{-x}}$. We analyzed this function by APCA over intervals $[-2, 2]$ and $[-8, 8]$ with two different choices for the pivot point b. As mentioned

above, in APCA two of the pivot points, a and c (x_0 and x_{m-1}), form the endpoints of the searched segment (a is fixed and c is moving to the right). From the computation point of view fixing a and b ($= x_1$) is acceptable. However, as the analysis revealed, putting b ($= x_{\frac{m}{2}}$ for even m) to the center of the searched segment may result in a smaller number of segments. Table shows the dependence of the segments' number on the choice of the pivot point b, interval and δ.

Table. The number of segments detected by APCA.

	$[-2,2]$ ($\delta = 0.02$)	$[-8,8]$ ($\delta = 0.0015$)
$b = x_1$	1	5
$b = x_{\frac{m}{2}}$	1	4

The computation of the APCA estimated ($\delta = 0.02, b = x_{\frac{m}{2}}$) cubic polynomial $y = 0.489812381x - 0.028332611x^3$ for the interval $[-2,2]$ is more than six times faster than the computation of the activation function. Although the computation of the activation function is only a part of the FFNN, it is not a negligible result when NN based computation takes several days. The computation time of the whole FFNN after replacement of the activation function with the given cubic polynomial requires by 5 percent less time.

6. CONCLUSION

In the first part of the paper a new representation of polynomials is given. Although the paper uses only cubic model, the representation enables to use approximation models of higher degree. The proof, based on the backward DPT, can be applied to the derivation of more complex approximation models that incorporate additional pivot points.

The second part is devoted to the neural networks and application of APCA. The combination of NN, APCA and splines enabled us to develop a novel approach to smoothing noisy data that consist of three steps: in the first step local estimations are computed by NN in several points, in the second one the APCA is used to reduce the number of the segments and in the third cubic splines are constructed. This combined method gave the most satisfied result for the given data among the methods considered in the paper.

ACKNOWLEDGEMENT

We thank A. Ďuricová and M. Rovňák for providing the data.
This work was partially supported by VEGA Grant 1/1006/04 9150 MŠ.

(Received March 23, 2006.)

REFERENCES

[1] N. D. Dikoussar: Function parametrization by using 4-point transforms. Comput. Phys. Comm. *99* (1997), 235–254.
[2] N. D. Dikoussar and Cs. Török: Automatic knot finding for piecewise-cubic approximation. Mat. Model. *T–18* (2006), 3, 23–40.
[3] D. Kahaner, C. Moler, and S. Nash: Numerical Methods and Software. Practice–Hall, Englewood Cliffs, N.J. 1989.
[4] S. Mallat: A Wavelet Tour of Signal Processing. Academic Press, New York 1999.
[5] M. Révayová and Cs. Török: Analysis of prediction with neural networks. In: Prastan 2004, Bratislava, pp. 85–93.
[6] B. D. Riplay: Pattern Recognition and Neural Networks. Cambridge University Press, Cambridge 1996.
[7] G. A. F. Seber: Linear Regression Analysis. Wiley, New York 1977.
[8] Cs. Török: 4-Point transforms and approximation. Comput. Phys. Comm. *125* (2000), 154–166.
[9] Cs. Török and N. D. Dikoussar: Approximation with discrete projective transformation. Comput. Math. Appl. *38* (1999), 211–220.
[10] Cs. Török: Visualization and data analysis in the MS.NET framework. In: Comm. JINR 2004, E10-2004-136, pp. 1–22.

Martina Révayová and Csaba Török, Department of Mathematics, Technical University of Košice, Vysokoškolská 4, 042 00 Košice. Slovak Republic.
e-mails: csaba.torok2@gmail.com, martina.revayova@gmail.com