NOVI SAD J. MATH.
VOL. 28, NO. 2, 1998, 31-35

RECURSIVE FORMULAE FOR PIECEWISE POLYNOMIAL APPROXIMATION OF DISCRETE FUNCTIONS

Miroslav Trajković, Mark Hedley
Department of Electrical Engineering, The University Sydney
NSW 2006, Australia

Abstract

To find local parameters of discrete 1D or 2D functions local polynomial approximation is often used. In this paper, we present a new fast algorithm for the recursive computation of approximating polynomials.

AMS Mathematics Subject Classification (1991):
Key words and phrases: Orthogonal Polynomials, Discrete functions, Image Processing, Histogram Analysis

Introduction : To analyze a discrete function $f(r)$ $(r = 0, 1, ..., N-1)$, at each point r it is convenient to approximate it over a symmetric window centered at r by a polynomial P_k^r, of order k.

If the size of the window is $2n+1$, then we can find P_k^r for each $r \in \{n, n+1, ..., N-1-n\}$. For convenience we shift $f(r)$ so that the point r about which we are approximating function is at the origin, so we define

$$f^r(i) = f(r+i).$$

We want to find the unique polynomial $P_k^r(i)$ that is the best approximation (in the mean squared error sense) of $f^r(i)$ over $i = (-n, -n+1, ..., n)$,

hence $P_k^r(i)$, which minimizes the functional

$$\sum_{i=-n}^{n} (f^r(i) - P_k^r(i))^2.$$

It can be done using discrete Chebyshev orthogonal polynomials (DCOP) [1] and $f^r(i)$ is approximated by

$$f^r(i) \approx \sum_{l=0}^{k} a_l^r Q_l(i) = P_k^r(i), \tag{1}$$

where $Q_l(i)$ is the corresponding DCOP and a_l^r are the coefficients computed for each r as

$$a_l^r = \frac{\sum_{i=-n}^{n} f^r(i) Q_l(i)}{\sum_{i=-n}^{n} Q_l^2(i)} = \frac{b_l^r}{||Q_l||^2}. \tag{2}$$

The equation (1) is very useful in practice, because it permits us to interpret $f(r)$ as a well-behaved function defined on the real line. It means that integrals and derivatives at any point can be easily estimated, which is useful in histogram analysis, for finding characteristic points such as knee, maximum and minimum. On the other hand, 2D extension of the equation (1) forms the basis of the so-called facet model [2] in image processing, used for the image surface description. The major drawback in this method is computational inefficiency. Namely, the coefficients a_l^r $(l = 0, 1, ..., k)$ are usually computed for each r separately, which is especially inefficient in image processing due to a large number of points. To overcome this problem, we propose an algorithm to compute the coefficients a_l^{r+1} recursively, using those (a_l^r) previously computed.

Recursive computation of the coefficients a_l^r (b_l^r) : Let us suppose that we have computed b_l^r and that we want to compute b_l^{r+1}. We have

$$\begin{aligned} b_l^{r+1} &= \sum_{i=-n}^{n} f^{r+1}(i) Q_l(i) = \sum_{i=-n+1}^{n+1} f^r(i) Q_l(i-1) \\ &= f^r(n+1) Q_l(n) - f^r(-n) Q_l(-n) + \sum_{i=-n}^{n} f^r(i) Q_l(i-1) \end{aligned} \tag{3}$$

On the other hand, since $Q_l(i-1)$ is a polynomial of degree l, it can be written as a linear combination of the polynomials $Q_0(i), Q_1(i), ..., Q_l(i)$, i. e.

$$Q_l(i-1) = Q_l(i) - \sum_{j=0}^{l-1} \alpha_{lj} Q_j(i) \tag{4}$$

where α_{lj} are the constant coefficients. Using properties of orthogonality it can be easily shown that

$$\alpha_{lj} = \frac{\sum_{i=-n}^{n} Q_l(i-1)Q_j(i)}{||Q_j||^2} \tag{5}$$

Hence, the coefficients α_{lj} once computed can be stored in memory and considered as *a priori* known constants. Substituting (4) in (3), after short rearrangement we get

$$b_l^{r+1} = f^r(n+1)Q_l(n) - f^r(-n)Q_l(-n) + b_l^r - \sum_{j=0}^{l-1} \alpha_{lj} b_l^j \tag{6}$$

and a_l^{r+1} is simply computed from (2).

As an example, let $n = 4$ and $k = 2$; then, using (5) we have:

$$\alpha_{10} = 1, \alpha_{20} = 17/3, \alpha_{21} = 2,$$

and, using (6)

$$\begin{aligned} b_0^{r+1} &= b_0^r + f^r(n+1) - f^r(-n), \\ b_1^{r+1} &= b_1^r + 4f^r(n+1) + 4f^r(-n) - b_0^r, \\ b_2^{r+1} &= b_2^r + 28/3 f^r(n+1) - 28/3 f^r(-n) - 17/3 b_0^r - 2b_1^r. \end{aligned}$$

Note that using formula (2), the numbers of multiplication (N_m) and addition (N_a) are independent of l, namely for every b_l^r, $N_m = 2n+1$ and $N_a = 2n$. Using (6), however, N_m and N_a both depend of l and we have $N_m = N_a = 2(l+1)$. Hence, all but last coefficient ($l = 2n$) are computed faster using (6) than (2). The above fact is especially true for small k which is mostly used in practice (k us rarely greater than 3, and $k = 2$ is frequently used).

To illustrate performances of the algorithm, we have computed total numbers of multiplications (T_m) and additions (T_a) using (2) and (6) respectively. Multiplication speed-up is defined as $S_m = T_m(nonrecursive)/T_m(recursive)$; a similar formula holds for addition speed-up S_a. After straightforward arithmetic it can be shown that the following formulae hold:

$$S_m = \frac{4n+2}{k+4} \quad \text{and} \quad S_a = \frac{4n}{k+4} \tag{7}$$

The values of S_m and S_a for $n = 1, 2, ..., 5$ and $k = 0, 1, 2, 3$ are shown in Table 1. (Note that for $n = 1$ and $k = 3$ result is not available because we can not approximate functions given at three points ($2n + 1 = 3$) by a third-order polynomial (i.e. $b_3^r = 0$)).

	n				
k	1	2	3	4	5
0	1.50/1.00	2.50/2.00	3.50/3.00	4.50/4.00	5.50/5.00
1	1.20/0.80	2.00/1.60	2.80/2.40	3.60/3.20	4.40/4.00
2	1.00/0.66	1.66/1.33	2.33/2.00	3.00/2.66	3.66/3.33
3	NA	1.43/1.14	2.00/1.71	2.57/2.28	3.14/2.86

Table 1. Multiplication and addition speed-up (S_m/S_a) for recursive algorithm.

Remark: Using symmetrical properties of DCOP both (6) and (2) can be computed faster, and new speed-ups computed, as shown in Table 2.

	n				
k	1	2	3	4	5
0	1.00/1.00	1.00/2.00	1.00/3.00	1.00/4.00	1.00/5.00
1	1.00/0.60	2.00/1.40	3.00/2.20	4.00/3.00	5.00/3.80
2	0.50/0.56	1.00/1.22	1.50/1.89	2.00/2.56	2.50/3.22
3	NA	0.75/1.00	1.12/1.57	1.50/2.14	1.88/2.72

Table 2. Multiplication and addition speed-up (S_m/S_a) for improved algorithms.

Table 2, however, shows only the lower bound of speed-up, since just formula (2) is optimally computed (can not be computed faster), while (6) is only improved - not necessarily optimized. It can be noted that for small n speed-up is also small. This is because for those values of n (1 or 2) the number of computation is already small and recursivity can not help very much. Full power of the algorithm is exploited for high values of n when the speed-up is linearly proportional to n.

Conclusion : In this paper we have presented an algorithm for recursive piecewise approximation of discrete functions by polynomials. The algorithm is much faster than the standard algorithm, especially for approximation by polynomials up to cubic.

References

[1] Hildebrand, F.B., Introduction to Numerical Analysis, McGraw Hill, 1974.

[2] Harralick R., Digital step edges from zero crossing of second directional derivatives, IEEE Trans. PAMI 6 (1984), 58-68.

Received by the editors July 10, 1997.