Annales UMCS Informatica AI 6 (2007) 95-102

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Fast algorithms for polynomial evaluation and differentiation at special knots

Joanna Kapusta*, Ryszard Smarzewski**

Department of Mathematics and Computer Sciences, The John Paul II Catholic University of Lublin, Konstantynów 1H, 20-708 Lublin, Poland

Abstract

We present fast evaluating and differentiating algorithms for the Hermite interpolating polynomials with the knots of multiplicity 2, which are generated dynamically in a field $K=(K,+,\cdot)$ by the recurrent formula of the form

$$x_i=\alpha x_{i-1}+\beta \quad (i=1,2,..,n-1;\ x_0=\gamma).$$

As in the case of Lagrange-Newton interpolating algorithms, the running time of these algorithms is $C(n)+O(n)$ base operations from the field *K*, where $C(n)=O(n\log n)$ denotes the time needed to compute the wrapped convolution in K^n.

1. Introduction and preliminaries

Let $K=(K,+,\cdot)$ be a field and let x_i $(i=0,1,..,n-1)$ be n pairwise distinct points in the field *K*. Additionally, let the values

$$y_i,z_i\in K \quad (i=0,1,...,n-1)$$

be given. Then there exists [1] a unique polynomial

$$p(x)=\sum_{i=0}^{n-1}y_i c_i(x)+\sum_{i=0}^{n-1}z_i d_i(x) \tag{1}$$

in the space $K_{2n-1}[x]$ of all polynomials of degree less than 2*n*, which is determined by the following Hermite interpolating conditions

$$p(x_i)=y_i \text{ and } p'(x_i)=z_i \ (i=0,1,...,n-1). \tag{2}$$

Moreover, the basic polynomials $c_i(x)$ and $d_i(x)$ are given by the following formulae

E-mail address: jkapusta@kul.lublin.pl
**E-mail address*: rsmax@kul.lublin.pl

$$\begin{aligned} c_i(x) &= \left[1-(x-x_i)\left(l_i(x_i)+l_i'(x_i)\right)\right]l_i^2(x), \\ d_i(x) &= (x-x_i)l_i^2(x) \qquad (i=0,1,2,...,n-1) \end{aligned} \tag{3}$$

with

$$l_i(x)=\prod_{k=0,k\neq i}^{n-1}\frac{x-x_k}{x_i-x_k}$$

In numerical computations [1,2], the Newton – Hermite interpolating formula

$$p(x)=\sum_{k=0}^{n-1}\left[f_{2k}+f_{2k+1}(x-x_k)\right]\prod_{\nu=0}^{k-1}(x-x_\nu)^2 \tag{4}$$

is preferred, where the generalized divided differences f_i ($i = 0,1,\ldots,2n-1$) are computed by the usual recurrent formulae, which requires $O(n^2)$ base operations from the field K. A faster algorithm is announced in Section 3 without proofs.

2. Fast evaluation and differentiation of polynomials

Let x_i ($i = 0,1,\ldots,n-1$) be n pairwise distinct points from the field K generated dynamically by the following recurrent formula

$$x_i=\alpha x_{i-1}+\beta \quad (i=1,2,..,n-1;\ x_0=\gamma), \tag{5}$$

where $\alpha \neq 0$, β, γ are fixed in the field K. In this case, it is possible to present a fast algorithm for computation of values

$$y_i=p(x_i) \quad (i=0,1,...,n-1)$$

and derivatives

$$z_i=p'(x_i) \quad (i=0,1,...,n-1)$$

of Hermite interpolating polynomial

$$p(x)=\sum_{k=0}^{n-1}\left[g_k+h_k(x-x_k)\right]\prod_{\nu=0}^{k-1}(x-x_\nu)^2 \tag{6}$$

where coefficients g_k and h_k ($k = 0,1,\ldots,n-1$) are given. Of course, these coefficients are generalized divided differences

$$\begin{aligned} g_k &= p[x_0,x_0,...,x_{k-1},x_{k-1},x_k], \\ h_k &= p[x_0,x_0,...,x_{k-1},x_{k-1},x_k,x_k] \quad (k=0,1,..,n-1) \end{aligned}$$

of the polynomial $p(x)$.

Indeed, one can substitute x_i into formula (6) and differentiate (6) at $x = x_i$ to get

$$y_i=\sum_{k=0}^{i}g_k\prod_{\nu=0}^{k-1}(x_i-x_\nu)^2+\sum_{k=0}^{i-1}h_k(x_i-x_k)\prod_{\nu=0}^{k-1}(x_i-x_\nu)^2 \tag{7}$$

and

$$z_i = \sum_{k=0}^{i} h_k \prod_{\nu=0}^{k-1} (x_i - x_\nu)^2 + 2g_i \prod_{\nu=0}^{i-1} (x_i - x_\nu) \sum_{\nu=0}^{i-1} \prod_{\substack{\mu=0 \\ \mu\neq\nu}}^{i-1} (x_i - x_\mu)$$

$$+ \sum_{k=0}^{i-1} \left[g_k + h_k (x_i - x_k) \right] \left(2 \prod_{\nu=0}^{k-1} (x_i - x_\nu) \sum_{\nu=0}^{k-1} \prod_{\substack{\mu=0 \\ \mu\neq\nu}}^{k-1} (x_i - x_\mu) \right). \tag{8}$$

Next we rewrite formula (5) in the form

$$x_i = \alpha^i \gamma + \beta \left(\alpha^{i-1} + \alpha^{i-2} + \ldots + 1 \right). \tag{9}$$

By inserting x_i into formulae (7) and (8) we obtain

$$y_i = \sum_{k=0}^{i} g_k \left(\frac{q_k p_i}{p_{i-k}} \right)^2 + \sum_{k=0}^{i-1} h_k r_k t_{i-k-1} \left(\frac{q_k p_i}{p_{i-k}} \right)^2$$

and

$$z_i = \sum_{k=0}^{i} h_k \left(\frac{q_k p_i}{p_{i-k}} \right)^2 + 2 q_i p_i \sum_{\mu=0}^{i-1} \frac{1}{r_\mu t_{i-\mu-1}}$$

$$+ \sum_{k=0}^{i-1} 2 \left[g_k + h_k r_k t_{i-k-1} \right] \left(\frac{q_k p_i}{p_{i-k}} \right)^2 \left(\sum_{\mu=0}^{i-1} \frac{1}{r_\mu t_{i-\mu-1}} - \frac{1}{r_k} \sum_{\mu=0}^{i-k-1} \frac{1}{r_\mu t_{i-k-1-\mu}} \right),$$

where

$$e = \gamma - \frac{\beta}{1-\alpha}, \quad r_j = \alpha^j, \quad t_j = e\left(\alpha^{j+1} - 1 \right), \tag{10}$$

$$q_j = \prod_{\nu=0}^{j-1} \alpha^\nu, \; p_j = \prod_{\nu=0}^{j-1} e\left(\alpha^{\nu+1} - 1 \right) \quad (j = 0,1,\ldots,n-1).$$

In these formulae, it is assumed that products are equal to 1 and sums are equal to 0 whenever their upper indices are smaller than the lower ones. As in the Lagrange-Newton interpolating algorithm [3], a computation of these formulae uses only the following six different vector operations in K^n:

- coordinatewise vector addition, subtraction, multiplication, division and multiplication by scalars,
- wrapped convolution defined as

$$a \otimes b = (c_0, c_1, .., c_{n-1}),$$

where

$$a = (a_0, a_1, \ldots, a_{n-1}), \; b = (b_0, b_1, \ldots, b_{n-1})$$

and

$$c_i = \sum_{k=0}^{i} a_k b_{i-k} \quad (i = 0,1,...,n-1).$$

Indeed, if we denote

$$\begin{gathered} g = (g_i)_0^{n-1},\ h = (f_i)_0^{n-1},\ d = (2p_i q_i w_i)_0^{n-1}, \\ y = (y_i)_0^{n-1},\ z = (z_i)_0^{n-1},\ r = (r_i)_0^{n-1},\ t = (t_i)_0^{n-1}, \\ p = (p_i)_0^{n-1},\ q = (q_i)_0^{n-1},\ w = (w_i)_0^{n-1}, \\ j = (1)_0^{n-1},\ v = (q_i^2)_0^{n-1},\ u = (p_i^2)_0^{n-1} \end{gathered} \tag{11}$$

then one can get

$$w = (j/r) \otimes (j/t)$$

and

$$\begin{gathered} y = \left[(g*v) \otimes (j/u) + (h*r*v) \otimes (t/u)\right]*u \\ z = \left\{(h*v) \otimes (j/u) + 2*w*\left[(g*v) \otimes u + (h*r*v) \otimes (t/u)\right] + \right. \\ \left. -2*\left[(g*v/r) \otimes (w/u) + (h*v) \otimes (t*w/u)\right]\right\}*u + d. \end{gathered}$$

Now we present the algorithm to compute the required values and derivatives. It uses two classes `KType` and `KTypeVector`, which should make it possible to perform operations in K and K^n.

Algorithm 1. Polynomial evaluation and differentiation at knots

$$x_i = ax_{i-1} + b \quad (i = 1,2,...,n-1;\ \ x_0 = c)$$

Input: A vectors $g = (g_0, g_1, ..., g_{n-1})$, $h = (h_0, h_1, ..., h_{n-1}) \in \mathrm{K}^n$ and three scalars $\alpha \neq 0$, β and γ in a field K.

Output: $y, z \in \mathrm{K}^n$.

1. Set $e \leftarrow c - b/(1-a)$, $p_0 \leftarrow 1$, $q_0 \leftarrow 1$, $r_0 \leftarrow 1$ and $v_0 \leftarrow 0$.
2. For k from 1 to $n-1$ do the following:
   2.1. $r_k \leftarrow r_{k-1} \cdot a$, $t_k \leftarrow (r_k - 1) \cdot e$,
   2.2. $p_k \leftarrow p_{k-1} \cdot r_{k-1}$, $q_k \leftarrow q_{k-1} \cdot t_{k-1}$.
3. Set $t_{n-1} \leftarrow (r_{n-1} \cdot a - 1) \cdot e$.
4. Compute $w \leftarrow (j/r \otimes j/t)$, $p2 \leftarrow p \cdot p$, $v \leftarrow q \cdot q$,
   $gv \leftarrow g \cdot v$, $hrv \leftarrow hv \cdot r$, $tu \leftarrow t/u$,
   $ju \leftarrow j/u$, $d \leftarrow 2 \cdot p \cdot q \cdot w$.
5. Compute $y \leftarrow (gv \otimes ju + hrv \otimes tu) \cdot u$.

6. Compute
$$z \leftarrow \left(hv \otimes ju + 2 \cdot w \cdot \left(gv \otimes u + hrv \otimes tu\right) - \right.$$
$$\left. -2 \cdot \left(\left(gv/r\right) \otimes \left(w/u\right) + hv \otimes \left(tu/w\right)\right)\right) \cdot u.$$
7. Return y, z.

Since the wrapped convolutions

$$conv\left(p,q\right) = p \otimes q\,,$$

with

$$p = \left(p_0, p_1, ..., p_{n-1}\right) \text{ and } q = \left(q_0, q_1, ..., q_{n-1}\right),$$

can be computed by an algorithm having a running time of $O(n\log n)$ base field operations in K [4], it follows that computation of values and derivates requires only $O(n\log n)$.

3. Fast computation of generalized divided differences

For the completeness, we include also a very short description of the inverse algorithm to Algorithm 1. Since the generalized divided differences f_{2k+1} ($k = 0,1,\ldots,n-1$) are coefficients at x^{2k+1} [1] in Hermite polynomials of degree $2k+1$ determined by the interpolating conditions

$$s\left(x_i\right) = y_i \text{ and } s'\left(x_i\right) = z_i \quad \left(i = 0,1,...,k\right),$$

it follows that

$$f_{2k+1} = \sum_{j=0}^{k}\left[\frac{z_j}{\prod\limits_{\nu=0,\nu\neq j}^{k}\left(x_j - x_\nu\right)^2} - \frac{2y_j\sum\limits_{\nu=0,\nu\neq j}^{k}\left(x_j - x_\nu\right)^{-1}}{\prod\limits_{\nu=0,\nu\neq j}^{k}\left(x_j - x_\nu\right)^2}\right]. \tag{12}$$

On the other hand, the generalized divided differences f_{2k} ($k = 0,1,\ldots,n-1$) are coefficients at x^{2k} [1] of Hermite polynomials of degree $2k$ determined by the interpolating conditions

$$w\left(x_i\right) = y_i\,,\ w'\left(x_i\right) = z_i \quad \left(i = 0,1,...,k-1\right) \text{ and } w\left(x_k\right) = y_k\,.$$

Hence we get

$$f_{2k} = \sum_{j=0}^{k}\left[-\frac{\sum\limits_{\nu=0,\nu\neq j}^{k}\left(x_j - x_\nu\right)^{-1}}{\prod\limits_{\mu=0,\mu\neq j}^{k-1}\left(x_j - x_\mu\right)\prod\limits_{\mu=0,\mu\neq j}^{k}\left(x_j - x_\mu\right)} - \frac{\sum\limits_{\nu=0,\nu\neq j}^{k-1}\left(x_j - x_\nu\right)^{-1}}{\prod\limits_{\mu=0,\mu\neq j}^{k}\left(x_j - x_\mu\right)\prod\limits_{\mu=0,\mu\neq j}^{k-1}\left(x_j - x_\mu\right)}\right] y_j \tag{13}$$
$$+\sum_{j=0}^{k-1}\frac{z_j}{\prod\limits_{\mu=0,\mu\neq j}^{k}\left(x_j - x_\mu\right)\prod\limits_{\mu=0,\mu\neq j}^{k-1}\left(x_j - x_\mu\right)} + \frac{y_k}{\prod\limits_{\mu=0}^{k-1}\left(x_k - x_\mu\right)^2}.$$

These formulae can be used to derive a $O(n\log n)$ – algorithm to compute generalized divided differences in the case when the interpolating knots are generated by formula (9). Now we present the algorithm to compute the required generalized divided differences in the Hermite interpolating formula (2). It uses two classes `KType` and `KTypeVector`, which make it possible to perform operations in K and $K^n K^n$. First we perform initial computation preparing to get generalized divided differences.

Algorithm 2. Computation of generalized divided differences with respect to knots

$$x_i = ax_{i-1} + b \quad (i = 1,2,...,n-1; \quad x_0 = c)$$

of multiplicity 2.

Input: A vectors $y = (y_0, y_1, ..., y_{n-1})$, $z = (z_0, z_1, ..., z_{n-1}) \in \mathrm{K}^n$ and three scalars $\alpha \neq 0$, β and γ in a field K.

Output: $g, h \in \mathrm{K}^n$.

1. Set $e \leftarrow c - b/(1-a)$, $p_0 \leftarrow 1$, $q_0 \leftarrow 1$, $r_0 \leftarrow 1$ and $v_0 \leftarrow 0$.
2. For k from 1 to $n-1$ do the following:
   2.1. $r_k \leftarrow r_{k-1} \cdot a$, $t_k \leftarrow (t_{k-1} - 1) \cdot e$, $q_k \leftarrow q_{k-1} \cdot t_{k-1}$.
   2.2. $p_k \leftarrow p_{k-1} \cdot r_{k-1}$ $v_k \leftarrow v_{k-1} - 1/t_{k-1}$.
3. Set $t_{n-1} \leftarrow (r_{n-1} \cdot a - 1) \cdot e$.
4. Compute $w \leftarrow (j/r \otimes j/t)$, $p2 \leftarrow p \cdot p$, $q2 \leftarrow q \cdot q$,
   $r2 \leftarrow r \cdot r$, $d \leftarrow y/(p2 \cdot q2)$, $u \leftarrow p2/q2$, $s \leftarrow u/t$,
   $rq2 \leftarrow r \cdot q2$, $yw \leftarrow y \cdot w$.
5. Compute $g \leftarrow \left(2 \cdot \left((yw/q2) \otimes u + (y/rq2) \otimes u\right) - (z/q2) \otimes u\right) \big/ p2.$
6. Compute
$$h \leftarrow 2 \cdot \left((yw/rq2) \otimes (s \cdot t) + (y/r2 \cdot p2) \otimes u\right) \big/ p2 -$$
$$-2 \cdot \left((y/r2 \cdot rq2) \otimes (u/t) + (z/rq2) \otimes s\right) \big/ p2 + d.$$
7. Return g, h.

The details of the proofs connected with this algorithm will be presented elsewhere.

It should be noticed that Algorithms 1 and 2 give a fast way to pass between the representations of a polynomial $p(x)$ in $K_{2n-1}[x]$ with respect to the Lagrange-Hermite base

$$c_0, d_0, c_1, d_1, .., c_{n-1}, d_{n-1}$$

and the Newton base

$$1,\ \ x-x_0,\ \ (x-x_0)^2,...,(x-x_0)^2...(x-x_{n-2})^2(x-x_{n-1}),$$
$$(x-x_0)^2...(x-x_{n-2})^2(x-x_{n-1})^2.$$

3. A complexity of algorithm

In this section we consider complexity of computation of Algorithm 1. The running time of this algorithm is $O(n\log n)$ base operations from the field K, while the running time of classical algorithm is $O(n^2)$. For example, if we choose n of order 2^{14}, then we save about 98 percent of base operations. More precisely, if $n = 2^{14}$ and if we use Algorithm 1, then the number of base operations for computing values and derivatives is approximately equal to 7.62×10^6, whereas the number of base operations in the classical algorithm is 5.37×10^8. Hence the number of saved base operations for Algorithm 1 equals 1.42% .

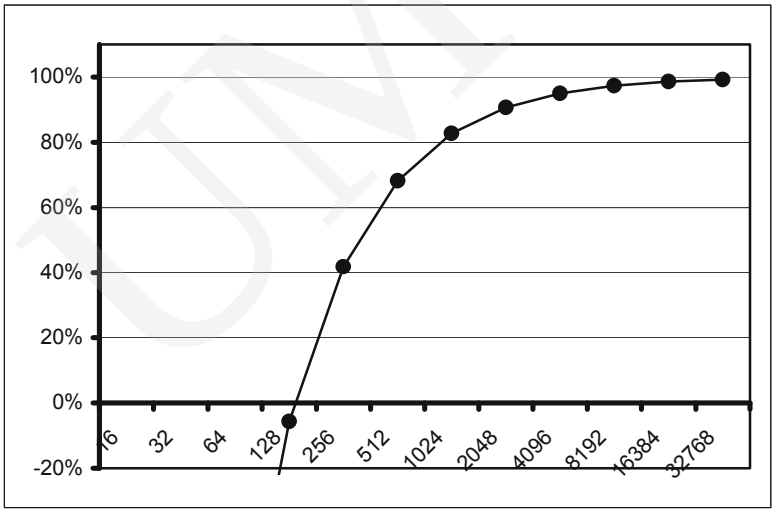

Fig. 1. The numbers of saved operations in percentage terms

The graph in Figure 1 shows the percentage of saved operations by Algorithm 1 in comparison with the classical $O(n^2)$ – algorithm for different values of n. The similar complexity results are also true for Algorithm 2.

4. Conclusions

In this paper, we have presented fast evaluating and differentiating algorithm for Hermite interpolating polynomial with n knots of multiplicity 2. These knots are generated dynamically in a field Kby the recurrent formula of the form

$$x_i = \alpha x_{i-1} + \beta \quad (i = 1,2,..,n-1;\ \ x_0 = \gamma).$$

The algorithm computes values and derivatives with the running time of $C(n) + O(n)$ base operations from the field K, where $C(n) = O(n\log n)$ denotes the complexity of computation of the wrapped convolution in K^n. On the other hand, the classical algorithm requires $O(n^2)$ of base operations in K. Numerical experiments show that this algorithm can be useful in practice, whenever n is sufficiently large. For example, such a situation occurs during the computation of shares and recovering keys in secret sharing schemes of Shamir type [5,6]. The similar results hold also for the inverse algorithm, which computes generalized divided differences.

References

[1] Ralston A., *A first Course in Numerical Analysis*. McGraw-Hill, New York, (1965).
[2] de Boor C., *A Practical Guide to Splines*. Springer-Verlag, New York, (2001).
[3] Smarzewski R., Kapusta J., *Fast Lagrange-Newton transformations*. Journal of Complexity, doi:10.1016/j.jco.2006.12.004, in press.
[4] Aho A.V., Hopcroft J.E., Ullman J.D., *The Design and Analysis of Computer Algorithms*. Addison-Wesley, London, (1974).
[5] Kapusta J., Smarzewski R., *Fast interpolating algorithms in cryptography*. Annales UMCS Informatica AI, 5 (2006) 37.
[6] Smarzewski R., Kapusta J., *Algorithms for multi-secret hierarchical sharing schemes of Shamir type*. Annales UMCS Informatica AI, 3 (2005) 65.