Computing 79, 33–52 (2007)
DOI 10.1007/s00607-006-0185-z
Printed in The Netherlands

Computing

A new efficient algorithm for polynomial interpolation

A. Smoktunowicz, **I. Wróbel**, and **P. Kosowski**, Warsaw

Received May 30, 2005; revised October 25, 2006
Published online: December 22, 2006

Abstract

A new backward stable algorithm (Algorithm 2) for polynomial interpolation based on the Lagrange and the Newton interpolation forms is proposed. It is shown that the Aitken algorithm and the scheme of the divided differences can be significantly less accurate than the proposed unconditionally stable Algorithm 2. Numerical examples that illustrate the advantage of a new algorithm are also given.

AMS Subject Classifications: 65D05, 65G50.

Keywords: Numerical stability, condition number, Lagrange form, Newton form, the divided differences, Aitken's algorithm.

1. Introduction

We propose a new backward stable algorithm for evaluation of the Lagrange representation of the interpolating polynomial $w_N(z) = \sum_{j=0}^{N} f_j \prod_{i=0, i\neq j}^{N} \frac{z-z_i}{z_j-z_i}$ and computing the Newton coefficients $c_0, \ldots, c_N$ of $w_N(z)$.

The problem of constructing efficient algorithms for polynomial interpolation involves matrices with special structure (cf. [3]). Computing the Newton coefficients of the interpolating polynomial is the first step of the Björck and Pereyra algorithm for solving Vandermonde systems of equations (cf. [2], [13]). We use a uniform approach and analyze also Aitken's algorithm of the evaluation of an interpolating polynomial. We assume that $z_0, z_1, \ldots, z_N$ are pairwise distinct data points and $f_0, f_1, \ldots, f_N$ are associated values (real or complex) of a function f. Then $w_N(z)$ interpolates $f(z)$ at points $z_0, z_1, \ldots, z_N$, i.e., $f(z_j) = f_j = w_N(z_j)$ for $j = 0, 1, \ldots, N$ and $w_N(z) = \sum_{j=0}^{N} c_j \prod_{i=0}^{j-1} (z - z_i)$.

It is known that there is only one interpolating polynomial $w_N(z)$. There exist different forms of $w_N(z)$, e.g., the Lagrange and the Newton forms and many algorithms for determining them. The problem of determining the Newton coefficients is intimately related with the problem of the evaluation of the Lagrange form of an interpolating polynomial, which can be realized by many algorithms. For these reasons we use a uniform approach and consider the following general problem.

Problem 1: For given $t \in \mathbb{C}, z_0, z_1, \ldots, z_N \in \mathbb{C}$ (distinct) and $f_0, f_1, \ldots, f_N$ compute

$$p_n(z;t) = \sum_{j=0}^{n} f_j \prod_{i=0,\ i\neq j}^{n} \frac{z - tz_i}{z_j - z_i}, \quad n = 0, 1, \ldots, N. \tag{1}$$

In the special case of $z = tz_j$ for some j we obtain $p_n(z,t) = f_j\, t^n$.

Notice that for $t = 1$ we have

$$p_n(z;1) = w_n(z) = \sum_{j=0}^{n} f_j \prod_{i=0,\ i\neq j}^{n} \frac{z - z_i}{z_j - z_i}, \quad n = 0, 1, \ldots, N \tag{2}$$

and for $t = 0$, $z = 1$, we get

$$p_n(1;0) = c_n = \sum_{j=0}^{n} f_j \prod_{i=0,\ i\neq j}^{n} \frac{1}{z_j - z_i}, \quad n = 0, 1, \ldots, N. \tag{3}$$

Usually the Newton coefficients are computed by the scheme of the divided differences. However, it is known that the reputation of this algorithm is not so good (cf. [10]). Its numerical properties depend on the distribution and ordering of the interpolation points (cf. [8]). Kiełbasiński ([7, pp. 51–53]) and Higham ([6, pp. 109–111]) proved that if the nodes are real and monotonically ordered, i.e., $z_0 < z_1 < \ldots < z_N$ or $z_0 > z_1 > \ldots > z_N$, then every Newton coefficient computed in floating point arithmetic (fl) is the exact divided difference for slightly perturbed values f_j (see Proposition 1), i.e., the algorithm is backward stable. Throughout the paper we will use the definition of the stability with respect to perturbations in the function values only, not the interpolation points.

Definition 1: *We say that an algorithm W of computing*

$$p_n(z;t) = \sum_{j=0}^{n} f_j \prod_{i=0,\ i\neq j}^{n} \frac{z - tz_i}{z_j - z_i}, \quad n = 0, 1, \ldots, N$$

is backward stable if the values $\tilde{p}_n(z;t)$ computed by W in floating point arithmetic satisfy

$$\tilde{p}_n(z;t) = \sum_{j=0}^{n} [f_j(1 + \delta_j^{(n)})] \prod_{i=0,\ i\neq j}^{n} \frac{z - tz_i}{z_j - z_i}, \quad |\delta_j^{(n)}| \leq \epsilon_M k_N, \tag{4}$$

where ϵ_M is the unit roundoff and k_N is a modest constant depending only on N.

The accuracy of the results obtained strongly depends on distribution of the interpolation points (see Sect. 4).

In practice it is very important to find good interpolation points. The "goodness" can be measured in many different ways, e.g., by the size of the associated Lebesgue constant (12), by distribution or by convergence for certain classes of functions.

As was pointed by Tal-Ezer in [17], the convergence in theory does not imply numerical accuracy.

In our tests (see Sect. 5), we used equally distributed points, Chebyshev points, fast Leja points and Leja ordered points. Fast Leja points have similar properties as Leja points and are much easier to compute. Leja points were introduced by Leja in [9], (see also [12], [1]). They are defined for a compact set K in the complex plane as follows.

Let $z_0 \in K$ be a point satisfying $|z_0| = \max_{z\in K} |z|$. The points $z_1, z_2, \ldots \in K$ are chosen in such a way that $\prod_{k=0}^{j-1} |z_j - z_k| = \max_{z\in K} \prod_{k=0}^{j-1} |z - z_k|$. Sometimes a compact set K is replaced by a discrete set of points. In this case we speak of Leja ordered points rather than Leja points.

The computation of Leja points requires the maximization of a product over a compact set K. To simplify the computations Baglama, Calvetti and Reichel [1] introduced fast Leja points. Here we will only describe how to determine the sequence of fast Leja points for an interval $[a; b]$. Suppose we have already computed k fast Leja points $z_1, z_2, \ldots, z_{k-1}$. To determine z_k we first define a set of candidate points $s_0, s_1, \ldots, s_{k-2}$. They are the bisection points between the existing fast Leja points. The next fast Leja point z_k is the one of the candidate points that satisfies $\prod_{l=0}^{k-1} |z_k - z_l| = \max_{j=0,1,\ldots,k-2} \prod_{l=0}^{k-1} |s_j - z_l|$. Then z_k is removed from the set of candidate points and two new candidate points are added, so that the sets of fast Leja points and candidate points interlace. The next fast Leja point z_{k+1} is determined analogously. The whole procedure is started with setting $z_0 = a$, $z_1 = b$. Then the only candidate point is $s_0 = (a + b)/2$. Thus $z_2 = (a + b)/2$ and a new set of candidate points is $\{(3a + b)/2, (a + 3b)/2\}$.

In Sect. 2, we review two standard methods, i.e., Aitken's interpolation algorithm and the scheme of the divided differences. Section 3 presents a new efficient algorithm (Algorithm 2) for computing (1). Error analysis of these algorithms is given in Sect. 7 (Appendix). The proposed Algorithm 2 is shown to be backward stable in the sense (4), with the constant k_N of order N (see Theorem 2), as opposed to the scheme of the divided differences and Aitken's algorithm. We also prove that these two algorithms are backward stable (with the constant k_N of order N, see Theorem 1) if the knots are real and monotonically ordered (cf. [7, pp. 51–53], [13]). Numerical tests in MATLAB included in Sect. 5 confirm theoretical results.

2. Existing algorithms

The simplest way to compute Newton coefficients (3) is the well known scheme of the divided differences, which can be visualized as follows:

$$c_j^{(0)} := f_j, \quad j = 0, 1, \ldots, N$$

$$
\begin{array}{lll}
z_0 & c_0^{(0)} \searrow & \\
 & & c_0^{(1)} \\
z_1 & c_1^{(0)} \nearrow & \searrow \\
 & & c_1^{(1)} \nearrow \quad c_0^{(2)} \\
z_2 & c_2^{(0)} \nearrow & \cdots \quad \cdots \qquad \searrow \\
 & \cdots & \qquad\qquad \nearrow \; c_0^{(N)}, \\
\cdots & \cdots & \\
 & & c_{N-2}^{(2)} \\
 & & c_{N-1}^{(1)} \nearrow \\
z_N & c_N^{(0)} \nearrow &
\end{array}
$$

where $c_j^{(n)}$ are the divided differences of n-th order. They are defined in the following way:

$$f(z_j; z_{j+1}; \ldots; z_{j+n}) = c_j^{(n)} = \frac{c_j^{(n-1)} - c_{j+1}^{(n-1)}}{z_j - z_{j+n}}$$

for $j = 0, 1, \ldots, N-n$, and $n = 1, 2, \ldots, N$.

To illustrate the idea of the scheme of the divided differences let us consider the interpolation based on three knots z_0, z_1 and z_2. The corresponding values are f_0, f_1 and f_2. Then the Newton coefficients are determined with help of the following table:

$$
\begin{array}{lll}
z_0 & f_0 \searrow & \\
 & & f(z_0; z_1) = \frac{f_1 - f_0}{z_1 - z_0} \\
z_1 & f_1 \nearrow & \searrow \\
 & \searrow & \qquad\qquad f(z_0; z_1; z_2) = \frac{f(z_1; z_2) - f(z_0; z_1)}{z_2 - z_0}. \\
 & & f(z_1; z_2) = \frac{f_2 - f_1}{z_2 - z_1} \nearrow \\
z_2 & f_2 \nearrow &
\end{array}
$$

The interpolating polynomial $w_2(z)$ has the following form:

$$w_2(z) = c_0^{(0)} + c_0^{(1)}(z - z_0) + c_0^{(2)}(z - z_0)(z - z_1),$$

where $c_0^{(0)} = f_0$, $c_0^{(1)} = f(z_0; z_1)$ and $c_0^{(2)} = f(z_0; z_1; z_2)$. It is easy to check that $w_2(z_k) = f_k$, $k = 0, 1, 2$.

Computing the value of the Lagrange form of the interpolating polynomial at a given point z (see formula (2)) can be realized by Aitken's algorithm (see [5, problem 8 on p. 289]).

As before, consider the case of three knots z_0, z_1 and z_2. We want to compute $w_0(z) = f(z_0) = f_0$, $w_1(z)$ and $w_2(z)$. We form the triangular table as above.

$$
\begin{array}{lll}
z_0 & f_0 \searrow & \\
 & & [z_0, z_1; z] \\
z_1 & f_1 \nearrow & \searrow \\
 & \searrow & \qquad\qquad [z_0, z_1, z_2; z], \\
 & & [z_1, z_2; z] \nearrow \\
z_2 & f_2 \nearrow &
\end{array}
$$

where

$$w_1(z) = [z_0, z_1; z] = \frac{\begin{vmatrix} z - z_0 & f_0 \\ z - z_1 & f_1 \end{vmatrix}}{z_1 - z_0}, \quad [z_1, z_2; z] = \frac{\begin{vmatrix} z - z_1 & f_1 \\ z - z_2 & f_2 \end{vmatrix}}{z_2 - z_1}$$

and

$$w_2(z) = [z_0, z_1, z_2; z] = \frac{\begin{vmatrix} z - z_0 & [z_0, z_1; z] \\ z - z_2 & [z_1, z_2; z] \end{vmatrix}}{z_2 - z_0}.$$

It is easy to verify that $w_1(z)$ is a polynomial of degree at most 1 and $w_1(z_0) = f_0$, $w_1(z_1) = f_1$. One can also easily check that $w_2(z_0) = [z_0, z_1, z_2; z_0] = f_0$, $w_2(z_1) = [z_0, z_1, z_2; z_1] = f_1$ and $w_2(z_2) = [z_0, z_1, z_2; z_2] = f_2$.

The following Algorithm 1 ([13]) can be used for computing both Newton coefficients and the value of the interpolating polynomial, depending on z and t. For $t = 0$ and $z = 1$ the algorithm reduces to the divided differences scheme, and for $t = 1$ it is Aitken's algorithm. For all other choices of z and t Algorithm 1 is a general way of evaluating (1).

Algorithm 1: For given (real or complex) z_n and f_n, t and z this algorithm computes

$$p_n(z; t) = \sum_{j=0}^{n} f_j \prod_{i=0,\ i\neq j}^{n} \frac{z - tz_i}{z_j - z_i}, \quad n = 0, 1, \ldots, N.$$

Choose t and z
$c_j^{(0)} := f_j, \quad j = 0, 1, \ldots, N$
for $n = 1, 2, \ldots, N$
  for $k = 0, 1, \ldots, N - n$
    $c_k^{(n)} := \frac{(z - tz_{n+k})\, c_k^{(n-1)} - (z - tz_k)\, c_{k+1}^{(n-1)}}{z_k - z_{k+n}}$ (5)
  end
  $p_n(z; t) := c_0^{(n)}$
end

Then

$$c_k^{(n)} = \sum_{j=k}^{k+n} f_j \prod_{i=k,\ i\neq j}^{k+n} \frac{z - tz_i}{z_j - z_i}, \quad n = 0, 1, \ldots, N, \quad k = 0, 1, \ldots, N - n.$$

It means that for $t = 1$, $c_k^{(n)}$ is the Lagrange representation of the interpolating polynomial based on the knots $z_k, z_{k+1}, \ldots, z_{k+n}$ and for $z = 1$, $t = 0$, $c_k^{(n)}$ is the divided difference of n-th order based on the knots $z_k, \ldots, z_{k+n}$. Sometimes in the

literature (see [5, problem 8 on p. 289]) one can find the formula (5) written in the following form:

$$c_k^{(n)} = \frac{\begin{vmatrix} z - tz_{n+k} & c_{k+1}^{(n-1)} \\ z - tz_k & c_k^{(n-1)} \end{vmatrix}}{z_k - z_{k+n}}.$$

The complexity of the divided differences and Aitken's algorithms is $N^2/2$ and $3N^2/2$ complex multiplications, respectively. Algorithm 1 requires $5N^2/2$ complex multiplications.

3. New algorithm

In this section, we propose a new algorithm for computing (1). The idea is a natural extension of the approach proposed by Werner in [18], and Rack and Reimer in [11]. They considered only the problem of computing a value of an interpolating polynomial. Our algorithm covers also the case of evaluating the Newton coefficients. It is based on evaluation of

$$p_n(z;t) = \prod_{j=0}^{n}(z - tz_j) \sum_{j=0}^{n} \frac{f_j}{(z - tz_j)\, w_j^{(n)}}, \quad w_j^{(n)} = \prod_{i=0,\ i\neq j}^{n} (z_j - z_i).$$

We develop an efficient way of computing $w_j^{(n)}$, different from what was proposed (only for $t = 1$) in [18], [11]. The detailed error analysis of Algorithm 2 is given in the Appendix (Sect. 7).

First, let us explain the idea of this algorithm in the case of three knots z_0, z_1 and z_2. The corresponding function values are f_0, f_1 and f_2. We will compute the Newton coefficients $c_0^{(0)}$, $c_0^{(1)}$ and $c_0^{(2)}$.

First set

$$c_0^{(0)} = f_0.$$

Then compute $c_0^{(1)}$ and $c_0^{(2)}$ in the following way:

$$c_0^{(1)} = \left[\frac{f_0}{z_0 - z_1}\right] + \left[\frac{f_1}{z_1 - z_0}\right],$$

$$c_0^{(2)} = \frac{\left[\frac{f_0}{z_0 - z_1}\right]}{z_0 - z_2} + \frac{\left[\frac{f_1}{z_1 - z_0}\right]}{z_1 - z_2} + \frac{f_2}{(-1)^2(z_0 - z_2)(z_1 - z_2)}.$$

Notice that to evaluate $c_0^{(2)}$ we use the values we already have from the previous step, i.e., from computing $c_0^{(1)}$. These are the ones in brackets.

Now let us turn our attention to the general case of $N+1$ interpolation knots and the general Problem 1.

For $z \neq tz_j$, $j = 0, 1, \ldots, N$

$$\prod_{i=0,\ i\neq j}^{n} (z - tz_i) = \frac{A_n}{z - tz_j}, \quad n = 0, 1, \ldots, N,$$

where

$$A_n = (z - tz_0)(z - tz_1) \cdot \ldots \cdot (z - tz_n).$$

Thus

$$p_n(z; t) = A_n \sum_{j=0}^{n} \frac{f_j}{(z - tz_j)\, w_j^{(n)}}, \quad w_j^{(n)} = \prod_{i=0,\ i\neq j}^{n} (z_j - z_i). \tag{6}$$

Note that A_n has the following form:

$$A_n = \begin{cases} (z - z_0) \cdot \ldots \cdot (z - z_n) & \text{for} \quad t = 1, \\ 1 & \text{for} \quad t = 0,\ z = 1. \end{cases}$$

We observe that

$$w_j^{(0)} = 1, \quad w_j^{(n)} = w_j^{(n-1)}(z_j - z_n) \quad \text{for} \quad j = 0, 1, \ldots, n-1,$$
$$A_n = (z - tz_n)A_{n-1}, \quad n = 0, 1, \ldots, N, \quad \text{with} \quad A_{-1} = 1.$$

Then for $n = 0, 1, \ldots, N$ we have $p_n(z; t) = A_n\ B_n$ where

$$B_n = \sum_{j=0}^{n} b_j^{(n)}, \quad b_j^{(n)} = \frac{f_j}{(z - tz_j)\, w_j^{(n)}}, \quad j = 0, 1, \ldots, n. \tag{7}$$

We see that

$$b_j^{(0)} = \frac{f_j}{z - tz_j}, \quad j = 0, 1, \ldots, N \tag{8}$$

and

$$b_j^{(n)} = \frac{b_j^{(0)}}{w_j^{(n)}}, \quad j = 0, 1, \ldots, n.$$

Notice that if B_{n-1} is written in the form

$$B_{n-1} = b_0^{(n-1)} + b_1^{(n-1)} + \ldots + b_{n-1}^{(n-1)},$$

then we can rewrite B_n as follows:

$$B_n = \frac{b_0^{(n-1)}}{(z_0 - z_n)} + \frac{b_1^{(n-1)}}{(z_1 - z_n)} + \ldots + \frac{b_{n-1}^{(n-1)}}{(z_{n-1} - z_n)} + \frac{b_n^{(0)}}{(z_n - z_0)(z_n - z_1)\ldots(z_n - z_{n-1})}.$$

Furthermore, the formulae (6) and (7) allow us to write

$$b_j^{(n)} = \frac{b_j^{(n-1)}}{z_j - z_n}, \quad j = 0, 1, \ldots, n-1, \quad n = 1, 2, \ldots, N \tag{9}$$

and

$$b_n^{(n)} = \frac{b_n^{(0)}}{\prod_{j=0}^{n-1}(z_n - z_j)}. \tag{10}$$

Algorithm 2: For given (real or complex) z_n and f_n, t and $z \neq tz_j$ this algorithm computes

$$p_n(z;t) = \sum_{j=0}^{n} f_j \prod_{i=0,\ i\neq j}^{n} \frac{z - tz_i}{z_j - z_i}, \quad n = 0, 1, \ldots, N.$$

Choose t and z
$b_j^{(0)} := \frac{f_j}{z - tz_j}, \quad j = 0, 1, \ldots, N,$
$A_0 := z - tz_0$
$B_0 := b_0^{(0)}$
$p_0(z;t) := 0$
for $n = 1, 2, \ldots, N$
  $A_n := (z - tz_n)A_{n-1}$
  for $j = 0, 1, \ldots, n-1$
    $b_j^{(n)} := \frac{b_j^{(n-1)}}{z_j - z_n}$
  end
  $b_n^{(n)} := \frac{b_n^{(0)}}{\prod_{j=0}^{n-1}(z_n - z_j)}$
  $B_n := \sum_{j=0}^{n} b_j^{(n)}$
  $p_n(z;t) := A_n B_n$
end

The complexity of all algorithms, counted as the number of complex multiplications and divisions is presented in Table 1.

Table 1. The complexity of Algorithm 1 and Algorithm 2

	Algorithm 1		Algorithm 2
	Div. diff.	Aitken	
Newton coefficients	$N^2/2$	–	N^2
Value of interpolating polynomial	–	$3N^2/2$	N^2

Note that this algorithm can be implemented more efficiently than it's written now, namely $b_n^{(n)}$ can be computed along with $b_j^{(n)}$ in the inner loop (over j) for

$$b_n^{(n)} := \frac{(-1)^n\, b_n^{(0)}}{\prod_{j=0}^{n-1}(z_j - z_n)}.$$

We decided to write Algorithm 2 this way to make it more clear.

4. Practical error bounds

Recall the definition of backward stability of computing (1) (Definition 1). It is easy to check that (4) is equivalent to

$$|\tilde{p}_n(z;t) - p_n(z;t)| \le \epsilon_M k_N \sum_{j=0}^{n} |f_j| \prod_{i=0,\ i\neq j}^{n} \frac{|z - tz_i|}{|z_j - z_i|}, \quad n = 0, \ldots, N. \tag{11}$$

This formula in case of an algorithm for computing a value of an interpolating polynomial can be rewritten as follows:

$$|\tilde{p}_N(z) - p_N(z)| \le \epsilon_M k_N \sum_{j=0}^{N} |f_j| \prod_{i=0,\ i\neq j}^{N} \left|\frac{z - z_i}{z_j - z_i}\right|.$$

This yields

$$|\tilde{p}_N(z) - p_N(z)| \le \epsilon_M k_N \|f\|_\infty \Lambda_N(z),$$

where

$$\|f\|_\infty = \max_{0\le j\le N} |f_j|$$

and

$$\Lambda_N(z) = \sum_{j=0}^{N} \prod_{i=0,\ i\neq j}^{N} \left|\frac{z - z_i}{z_j - z_i}\right|$$

is the Lebesgue function. We can define the Lebesgue constant on the set S as follows:

$$\Lambda_N(S) = \max_{z\in S} \Lambda_N(z). \tag{12}$$

We assume that S consists of the interpolating points and some checkpoints. Note that $\Lambda_N(S) \ge 1$ for arbitrary data.

The tests we performed (see Sect. 5) confirm theoretical results about the stability of Algorithm 2. To check the behavior of our algorithm we computed the following stability factor:

$$E_N = \max_{z\in S} \frac{|\tilde{p}_N(z) - p_N(z)|}{\epsilon_M \sum_{j=0}^{N} |f_j| \prod_{i=0,\ i\neq j}^{N} \left|\frac{z-z_i}{z_j-z_i}\right|}. \tag{13}$$

The algorithm is stable if E_N is small.

Our analysis in the Appendix (Sect. 7) shows that in the case of Algorithm 2, $E_N = \mathcal{O}(N)$ and, moreover, it does not depend on the choice of the interpolating points.

To compare the results given by our algorithm and the classical Aitken's algorithm we computed also the relative error

$$Err(f) = \frac{\max_{z\in S} |f(z) - \tilde{p}_N(z)|}{\|f\|_\infty}. \tag{14}$$

Clearly, for a stable algorithm $Err(p_N) \leq \epsilon_M k_N \Lambda_N(S)$.

Now let us turn to the problem of computing the Newton coefficients. According to (3) and (11) an algorithm for computing the Newton coefficients c_n, for $n = 0, 1, \ldots, N$, is numerically stable if

$$|\tilde{c}_n - c_n| \leq \epsilon_M k_N \sum_{j=0}^{n} |f_j| \prod_{i=0,\ i\neq j}^{n} \left|\frac{1}{z_j - z_i}\right| \quad \text{for} \quad n = 0, 1, \ldots, N,$$

where $\tilde{c}_n$ and c_n are the computed and exact Newton coefficients, respectively. This yields

$$|\tilde{c}_n - c_n| \leq \epsilon_M k_N \|f\|_\infty \sum_{j=0}^{n} \frac{1}{\prod_{i=0,\quad i\neq j}^{n} |z_j - z_i|} \quad \text{for} \quad n = 0, 1, \ldots, N,$$

and further

$$\frac{\max_{n=0,1,\ldots,N} |\tilde{c}_n - c_n|}{\|f\|_\infty} \leq \epsilon_M k_N \lambda_N,$$

where

$$\lambda_N = \max_{n=0,1,\ldots,N} \sum_{j=0}^{n} \frac{1}{\prod_{i=0,\quad i\neq j}^{n} |z_j - z_i|} \quad \text{for} \quad n = 0, 1, \ldots, N. \tag{15}$$

We will refer to λ_N as the Leja constant for the knots $z_0, z_1, \ldots, z_N$. Note that always $\lambda_N \geq 1$.

In our tests we computed two quantities, the relative error

$$\hat{Err}(f) = \frac{\max_{n=0,1,\ldots,N} |\tilde{c}_n - c_n|}{\|f\|_\infty} \tag{16}$$

and the stability factor

$$\hat{E}_N = \max_{n=0,1,\ldots,N} \frac{|\tilde{c}_n - c_n|}{\epsilon_M \sum_{j=0}^{n} |f_j| \prod_{i=0,\ i\neq j}^{n} \left|\frac{1}{z_j - z_i}\right|}. \tag{17}$$

The algorithm is stable if $\hat{E}_N$ is small. Moreover, if the interpolating points are well-distributed in the sense that the Leja constant λ_N is small, then the error $\hat{Err}(f)$ is of order ϵ_M.

Table 2. The error (13) for Aitken's algorithm and Algorithm 2, the function $f(z) = z^7$ and fast Leja points for the interval $[-2; 2]$

$N+1$	Aitken	Algorithm 2	$\Lambda_N(S)$
10	$1.4684e+01$	$2.2067e+00$	$5.1991e+00$
20	$5.2850e+01$	$2.6491e+00$	$6.0669e+00$
40	$1.2073e+02$	$3.6984e+00$	$1.4912e+01$
80	$9.8481e+02$	$6.0701e+00$	$1.9356e+01$
120	$1.5617e+03$	$5.9228e+00$	$2.4055e+01$
160	$1.2996e+03$	$6.6234e+00$	$2.8713e+01$
200	$3.4967e+04$	$8.5341e+00$	$2.7620e+01$

5. Numerical experiments

To illustrate our results we present numerical experiments carried out in MATLAB with unit roundoff $\epsilon_M = 2^{-52} \approx 2.2e{-}16$.

We implemented all methods and performed many tests in order to investigate the behavior of these methods, i.e., to compare the errors generated by each of them. This paragraph contains the results of some of these tests.

The purpose of our tests was to investigate stability and accuracy of all algorithms.

Algorithm 2 is general, two special cases are computing the value of an interpolating polynomial and the Newton coefficients. These are the cases we considered in our tests. We compared the results given by Algorithm 2 with the ones given by classical Algorithm 1, which can be used for both problems, computing the divided differences and evaluating the interpolation polynomial by means of Aitken's algorithm. The results are collected in Sects. 5.1 and 5.2.

The accuracy of the results obtained strongly depends on distribution of interpolation points.

5.1. Evaluation of $\mathrm{p_N}$

This section contains the results of the tests of algorithms for the evaluation of an interpolating polynomial.

We present the results for three functions: $f(z) = z^7$, $f(z) = z + \frac{1}{z+2+\delta}$, with $\delta = 1e-4$, and $f(z) = \sqrt{1+z/2}$.

Table 2 contains the values of the error (13) for Aitken's algorithm and Algorithm 2 applied to the function $f(z) = z^7$. The interpolation was based on $N+1$ fast Leja points for the interval $[-2; 2]$. The number of knots is given in the first column. The last column contains the respective values of the Lebesgue constant (12) on the set S. In our tests S consists of the interpolation points and 400 equally spaced checkpoints from the interval $[-2+1/101; 2-1/101]$ (in order to make results more interesting).

Table 3. The error (13) for Aitken's algorithm and Algorithm 2, the function $f(z) = z + \frac{1}{z+2+\delta}$, $\delta = 1e{-}4$ and fast Leja points for the interval $[-2; 2]$

$N+1$	Aitken	Algorithm 2	$\Lambda_N(S)$
10	$4.0365e+01$	$3.9669e+00$	$5.1991e+00$
20	$1.8317e+01$	$5.3204e+00$	$6.0669e+00$
40	$8.6565e+02$	$6.7869e+00$	$1.4912e+01$
80	$5.3382e+02$	$6.4409e+00$	$1.9356e+01$
120	$2.0979e+03$	$6.8509e+00$	$2.4055e+01$
160	$1.9911e+03$	$7.9894e+00$	$2.8713e+01$
200	$9.8425e+03$	$8.5941e+00$	$2.7620e+01$

Table 4. The error (13) for Aitken's algorithm and Algorithm 2, the function $f(z) = \sqrt{1 + z/2}$ and fast Leja points for the interval $[-2; 2]$

$N+1$	Aitken	Algorithm 2	$\Lambda_N(S)$
10	$1.4551e+01$	$2.4027e+00$	$5.1991e+00$
20	$7.3547e+01$	$3.1419e+00$	$6.0669e+00$
40	$3.8038e+02$	$5.6180e+00$	$1.4912e+01$
80	$4.6200e+02$	$8.0169e+00$	$1.9356e+01$
120	$1.9227e+03$	$9.1835e+00$	$2.4055e+01$
160	$2.9968e+03$	$1.1621e+01$	$2.8713e+01$
200	$5.0612e+04$	$1.0337e+01$	$2.7620e+01$

Tables 3 and 4 contain analogous results for the functions $f(z) = z + \frac{1}{z+2+\delta}$, $\delta = 1e-4$ and $f(z) = \sqrt{1 + z/2}$, respectively.

For the sake of a better demonstration of the advantage of Algorithm 2 we present tests performed for interpolation points that are not monotonically ordered. In the case of monotone ordering the constant L in Theorem 1 is equal to 1 and Aitken's algorithm is backward stable. If the knots are monotonically ordered, e.g., uniformly distributed points or Chebyshev points, the results given by both algorithms were comparable and we do not present them here.

5.2. Newton coefficients

Here we present the results of tests of algorithms for computing the Newton coefficients.

The "exact" values of the Newton coefficients c_n were obtained by implementing Algorithm 2 in high precision using the VPA (Variable Precision Arithmetic) function from MATLAB's Symbolic Math Toolbox.

Tables 5 and 6 contain the values of the errors (16) and (17) for the divided differences algorithm and Algorithm 2 applied to the function $f(z) = z + \frac{1}{z+2+\delta}$, $\delta = 1e-4$. The interpolation was based on fast Leja points. The number of knots is given in the first column. The last column of Table 5 contains the respective values of the Leja constant (15).

Table 5. The relative error (16) for the divided differences algorithm and Algorithm 2, the function $f(z) = z + \frac{1}{z+2+\delta}$, $\delta = 1e-4$ and fast Leja points for the interval $[-2; 2]$

$N+1$	Algorithm 1	Algorithm 2	λ_N
10	$1.9145e-17$	$1.6209e-17$	$1.1111e+00$
20	$7.3086e-17$	$1.6209e-17$	$1.1363e+00$
40	$1.6677e-16$	$1.6209e-17$	$1.4260e+00$
80	$7.9042e-16$	$1.6209e-17$	$1.5201e+00$
120	$7.9042e-16$	$1.6209e-17$	$1.6900e+00$
160	$7.9042e-16$	$1.6209e-17$	$1.8902e+00$
200	$7.9042e-16$	$1.6209e-17$	$2.1290e+00$

Table 6. The stability factor (17) for the divided differences algorithm and Algorithm 2, the function $f(z) = z + \frac{1}{z+2+\delta}$, $\delta = 1e-4$ and fast Leja points for the interval $[-2; 2]$

$N+1$	Algorithm 1	Algorithm 2
10	$6.4894e+00$	$0.8673e+00$
20	$7.3564e+01$	$1.1807e+00$
40	$2.7582e+02$	$3.2605e+00$
80	$1.5856e+04$	$4.5205e+00$
120	$1.5856e+04$	$4.5205e+00$
160	$1.5856e+04$	$4.5205e+00$
200	$1.5856e+04$	$4.5205e+00$

Table 7. The relative error (16) for the divided differences algorithm and Algorithm 2, the function $f(z) = 1/(1+1000z^2)$ and fast Leja points for the interval $[-1; 1]$

$N+1$	Algorithm 1	Algorithm 2	λ_N
10	$1.5756e-14$	$2.8138e-15$	$3.6625e+02$
20	$3.1294e-11$	$2.1900e-12$	$4.0326e+05$
40	$3.8458e-05$	$2.3790e-06$	$4.1664e+11$
80	$3.2763e+07$	$3.4030e+06$	$6.7153e+23$

So far we have concentrated only on interpolation points from the interval $[-2; 2]$.

Now let us consider the function $f(z) = 1/(1+1000z^2)$ and fast Leja points from the interval $[-1; 1]$. The results are collected in Tables 7 and 8. The last column of Table 7 contains the respective values of the Leja constant (15).

Although Algorithm 2 still gives better results than the divided differences algorithm, both algorithms generate big errors, especially for larger numbers of knots. This is mainly due to the choice of the interpolation interval. By substituting $w = 2z$, let us change it into $[-2; 2]$ and compare the results. Now the interpolated function is $g(w) = 1/(1+250w^2)$. The interpolation points are still fast Leja points. Tables 9 and 10 contain the respective results for the new data. The value of the Lebesgue constant is contained in the last column of Table 9.

Table 8. The stability factor (17) for the divided differences algorithm and Algorithm 2, the function $f(z) = 1/(1 + 1000z^2)$ and fast Leja points for the interval $[-1; 1]$

$N+1$	Algorithm 1	Algorithm 2
10	$1.8292e+00$	$0.5000e+00$
20	$2.6135e+01$	$1.4096e+00$
40	$5.2650e+01$	$2.0316e+00$
80	$1.7172e+02$	$2.0316e+00$

Table 9. The relative error (16) for the divided differences algorithm and Algorithm 2, the function $g(w) = 1/(1 + 250w^2)$ and fast Leja points for the interval $[-2; 2]$

$N+1$	Algorithm 1	Algorithm 2	λ_N
10	$4.3012e-17$	$2.6116e-17$	$1.1111e+00$
20	$1.8877e-16$	$3.3793e-17$	$1.1363e+00$
40	$2.3896e-16$	$4.6603e-17$	$1.4260e+00$
80	$1.5555e-15$	$6.5534e-17$	$1.5201e+00$

Table 10. The stability factor (17) for the divided differences algorithm and Algorithm 2, the function $g(w) = 1/(1 + 250w^2)$ and fast Leja points for the interval $[-2; 2]$

$N+1$	Algorithm 1	Algorithm 2
10	$1.8292e+00$	$0.5000e+00$
20	$2.6135e+01$	$1.4096e+00$
40	$5.2650e+01$	$2.0316e+00$
80	$1.7172e+02$	$2.0316e+00$

Table 11. The relative error (16) for the divided differences algorithm and Algorithm 2, the function $g(w) = 1/(1 + 250w^2)$ and equidistant points for the interval $[-1; 1]$

$N+1$	Algorithm 1	Algorithm 2	λ_N
20	$1.6720e-10$	$3.3930e-10$	$1.6264e+07$
40	$0.0219e+00$	$0.2115e+00$	$5.5199e+15$

Now we present results for equidistant interpolating points from the interval $[-1; 1]$. The interpolated function is $g(w) = 1/(1 + 250w^2)$. The respective values of the relative error (16) and the stability factor (17) are collected in Tables 11 and 12. The last column of Table 11 contains values of the Leja constant (15).

Algorithm 2 is especially attractive for complex data. For example, for the function $f(z) = z^7$ and interpolation knots $1+i, -1+i, 0, 1, 10i, 7+3i, 2-i, 2+i, i, 7-3i$ we get the following results. The Leja constant $\lambda_9 = 1.4589$, the relative errors (16) for Algorithms 1 and 2 are equal $1.1127e-19$ and $1.6653e-23$, respectively. The respective values of the stability factor (17) for these algorithms are $2.8389e+03$ and 0.2358.

Table 12. The stability factor (17) for the divided differences algorithm and Algorithm 2, the function $g(w) = 1/(1 + 250w^2)$ and equidistant points for the interval $[-1; 1]$

$N+1$	Algorithm 1	Algorithm 2
20	0.3099e+00	0.3978e+00
40	0.3123e+00	0.5315e+00

Some of the examples presented here were inspired by works of Reichel [12], Tal-Ezer [17], and Baglama, Calvetti and Reichel [1].

6. Conclusions

We have shown that our algorithm (Algorithm 2) for polynomial interpolation is backward stable with respect to perturbations in the function values, as opposed to the scheme of the divided differences. Our new algorithm does not require any special ordering of interpolation points. Moreover, it is stable for any choice of interpolating knots.

7. Appendix: Error analysis of algorithms

We consider complex arithmetic (cfl) (cf. [16]) implemented using real arithmetic (fl) with machine unit ϵ_M. We assume that for $x, y \in \mathbb{R}$ we have

$$fl(x \pm y) = (x \pm y)(1 + \delta), \quad |\delta| \leq \epsilon_M. \tag{18}$$

Then

$$cfl(x \odot y) = (x \odot y)(1 + \delta), \quad |\delta| \leq c_{\odot}\,\epsilon_M \quad \text{for } x, y \in \mathbb{C}, \tag{19}$$

where

$$c_{\odot} = \begin{cases} 1 & \text{for } \odot \in \{+, -\}, \\ 1 + \sqrt{2} & \text{for } \odot = *, \\ 4 + \sqrt{2} & \text{for } \odot = /. \end{cases} \tag{20}$$

The constant $c_{\odot}$ is not significant, it depends on the implementation of floating point operations. In fact, the bounds of errors in Algorithms 1 and 2 are of the same form for real and complex arithmetic, only the constants are increased appropriately.

Our error analysis is uniform, it includes both real and complex cases, i.e., the data z_j, f_j, z and t can be either real or complex.

The values computed in fl or cfl will be marked with tilde in order to distinguish them from the exact ones.

7.1. Error analysis of Algorithm 1

Theorem 1: *Assume that z_j, $f_j, \in \mathbb{C}$ for $j = 0, 1, \ldots, N$ and z, $t \in \mathbb{C}$ are exactly representable in cfl and*

$$cfl(tz_j) = tz_j, \quad j = 0, 1, \ldots, N. \tag{21}$$

Then the values $\tilde{c}_k^{(n)}$ for $n = 0, 1, \ldots N$, $k = 0, 1, \ldots, N - n$ computed in cfl by Algorithm 1 satisfy

$$\tilde{c}_k^{(n)} = \sum_{j=k}^{k+n} [f_j(1 + \delta_{k,j}^{(n)})] \prod_{i=k,\ i\neq j}^{k+n} \frac{z - tz_i}{z_j - z_i}, \quad |\delta_{k,j}^{(n)}| \leq \epsilon_M n\, d\, L^{n-1} + \mathcal{O}(\epsilon_M^2), \tag{22}$$

where

$$d = \begin{cases} 5 & \text{in the real case,} \\ 8 + 2\sqrt{2} & \text{in the complex case} \end{cases}$$

and

$$L = \max_{0\leq i<j<k\leq N} \frac{|z_i - z_j| + |z_j - z_k|}{|z_i - z_k|}. \tag{23}$$

Proof: By induction on n. It follows from (5) and (18)–(20) that for $n = 0, 1, \ldots, N$, $k = 0, 1, \ldots, N - n$ the computed quantities satisfy

$$\tilde{c}_k^{(n)} := \frac{(1 + \alpha_k^{(n)})(z - tz_{n+k})\tilde{c}_k^{(n-1)} - (1 + \beta_k^{(n)})(z - tz_k)\tilde{c}_{k+1}^{(n-1)}}{z_k - z_{k+n}},$$

where

$$|\alpha_k^{(n)}|,\ |\beta_k^{(n)}| \leq \epsilon_M\, d + \mathcal{O}(\epsilon_M^2).$$

It is clear that for $n = 1$ the result is immediate, since

$$\tilde{c}_k^{(1)} = f_k\,(1 + \alpha_k^{(1)})\frac{z - tz_{k+1}}{z_k - z_{k+1}} + f_{k+1}\,(1 + \beta_k^{(1)})\frac{z - tz_k}{z_{k+1} - z_k},$$

so we can take $\delta_{k,k}^{(1)} = \alpha_k^{(1)}$ and $\delta_{k,k+1}^{(1)} = \beta_k^{(1)}$, $k = 0, 1, \ldots, N - 1$.

Now assume that (22) is true for $n - 1$. We will prove that it holds also for n. By assumption for $n - 1$ we get:

$$(z - tz_{n+k})\,\tilde{c}_k^{(n-1)} = \sum_{j=k}^{k+n-1} [f_j(1 + \delta_{k,j}^{(n-1)})] \prod_{i=k,\ i\neq j}^{k+n} \frac{z - tz_i}{z_j - z_i}\,(z_j - z_{k+n}), \tag{24}$$

$$(z - tz_k)\,\tilde{c}_{k+1}^{(n-1)} = \sum_{j=k+1}^{k+n} [f_j(1 + \delta_{k+1,j}^{(n-1)})] \prod_{i=k,\ i\neq j}^{k+n} \frac{z - tz_i}{z_j - z_i}\,(z_j - z_k). \tag{25}$$

To simplify the notation let us define

$$1+\phi_{k,j}^{(n)} = (1+\alpha_k^{(n)})(1+\delta_{k,j}^{(n-1)})$$

and

$$1+\psi_{k,j}^{(n)} = (1+\beta_k^{(n)})(1+\delta_{k+1,j}^{(n-1)}),$$

bounded as follows:

$$|\phi_{k,j}^{(n)}|,\ |\psi_{k,j}^{(n)}| \le \epsilon_M d\,(1+(n-1)L^{n-2}) + \mathcal{O}(\epsilon_M^2). \tag{26}$$

Subtraction of (24) from (25) leads to

$$\tilde{c}_k^{(n)} = \sum_{j=k}^{k+n} f_j(1+\delta_{k,j}^{(n)}) \prod_{i=k, i\neq j}^{k+n} \frac{z-tz_i}{z_j-z_i},$$

where

$$1+\delta_{k,j}^{(n)} = \frac{(1+\phi_{k,j}^{(n)})(z_j-z_{k+n}) + (1+\psi_{k,j}^{(n)})(z_k-z_j)}{z_k-z_{k+n}},$$

for $j = k+1, \ldots, k+n-1$, and

$$1+\delta_{k,k}^{(n)} = 1+\phi_{k,k}^{(n)}, \quad 1+\delta_{k,k+n}^{(n)} = 1+\psi_{k,k+n}^{(n)}.$$

This leads to the estimation

$$|\delta_{k,j}^{(n)}| \le \max\{|\phi_{k,j}^{(n)}|,\ |\psi_{k,j}^{(n)}|\}\, L, \quad \text{for} \quad j = k, \ldots, k+n,$$

which, in combination with (26) and the fact that $L \ge 1$ (see (23)) results in $|\delta_{k,j}^{(n)}| \le \epsilon_M n d\, L^{n-1} + \mathcal{O}(\epsilon_M^2)$, which completes the proof. □

Proposition 1: *Assume that $z_j,\ f_j \in \mathbb{R}$ for $j = 0, 1, \ldots, N$ and $z, t \in \mathbb{R}$ are exactly representable in fl and* (21) *holds. If the knots z_j are monotonically ordered then the values $\tilde{p}_n(z;t) = \tilde{c}_0^{(n)}$ computed in fl by Algorithm 1 satisfy*

$$\tilde{p}_n(z;t) = \sum_{j=0}^{n}[f_j(1+\delta_j^{(n)})] \prod_{i=0,\ i\neq j}^{n} \frac{z-tz_i}{z_j-z_i}, \quad |\delta_j^{(n)}| \le \epsilon_M 5n + \mathcal{O}(\epsilon_M^2).$$

Proof: It is a direct consequence of Theorem 1 for $k = 0$. For monotonically ordered knots z_j the constant L in (23) is 1. □

Remark 1: *If $z_j \in \mathbb{C},\ z_j = a + t_j(b-a),\ j = 0, 1, \ldots, N,\ t_0 < t_1 < \ldots < t_N$, $a, b \in \mathbb{C}$, then $L = 1$ and Algorithm 1 is also numerically stable, i.e., the condition (4) holds with the constant $k_N \approx (8+2\sqrt{2})N$.*

In Sect. 5, we presented numerical experiments showing that Algorithm 1 is not always backward stable.

7.2. Error analysis of Algorithm 2

Theorem 2: *Assume that z_j, $f_j \in \mathbb{C}$, $z \neq t z_j$ for $j = 0, 1, \ldots, N$ and $z, t \in \mathbb{C}$ are exactly representable in cfl and* (21) *holds. Then the values $\tilde{p}_n(z; t)$ computed in cfl by Algorithm 2 satisfy*

$$\tilde{p}_n(z;t) = \sum_{j=0}^{n} [f_j(1+\delta_j^{(n)})] \prod_{i=0,\ i\neq j}^{n} \frac{z - t z_i}{z_j - z_i}, \quad |\delta_j^{(n)}| \leq \epsilon_M k_N + \mathcal{O}(\epsilon_M^2), \tag{27}$$

where

$$k_N = (N+1) \begin{cases} 5 & \text{in the real case,} \\ 8 + 2\sqrt{2} & \text{in the complex case.} \end{cases}$$

Proof: The proof is straightforward. Using (8)–(10) and (18)–(20) we obtain the following quantities, computed in fl or cfl

$$\tilde{b}_j^{(0)} = \frac{f_j}{z - t z_j}(1 + \Delta_j^{(0)}), \quad j = 0, 1, \ldots, N,$$

and for $n = 1, 2, \ldots, N$

$$\tilde{A}_n = A_n(1 + \alpha_n),$$

$$\tilde{b}_j^{(n)} = \frac{\tilde{b}_j^{(n-1)}}{z_j - z_n}(1 + \beta_j^{(n)}), \quad j = 0, 1, \ldots, n-1,$$

$$\tilde{b}_n^{(n)} = \frac{\tilde{b}_n^{(0)}}{\prod_{j=0}^{n-1}(z_n - z_j)}(1 + \gamma_n).$$

From (7), we obtain

$$\tilde{B}_n = \sum_{j=0}^{n} \tilde{b}_j^{(n)}(1 + \alpha_j^{(n)}), \quad |\alpha_j^{(n)}| \leq \epsilon_M n + \mathcal{O}(\epsilon_M^2),$$

$$\tilde{p}_n(z;t) = \tilde{A}_n \tilde{B}_n(1 + \delta_n),$$

where

$$|\Delta_j^{(0)}|,\ |\beta_j^{(n)}| \leq \epsilon_M \begin{cases} 2 & \text{(real case)} \\ 5 + \sqrt{2} & \text{(complex case)} \end{cases} + \mathcal{O}(\epsilon_M^2),$$

$$|\alpha_n| \leq \epsilon_M \begin{cases} 2n + 1 & \text{(real case)} \\ (2 + \sqrt{2})n + 1 & \text{(complex case)} \end{cases} + \mathcal{O}(\epsilon_M^2),$$

$$|\gamma_n| \leq \epsilon_M \begin{cases} 2n & \text{(real case)} \\ (2 + \sqrt{2})n + 3 & \text{(complex case)} \end{cases} + \mathcal{O}(\epsilon_M^2),$$

$$|\delta_n| \leq \epsilon_M \begin{cases} 1 & \text{(real case)} \\ 1 + \sqrt{2} & \text{(complex case)} \end{cases} + \mathcal{O}(\epsilon_M^2).$$

Now the formulae (9), (10) yield for all $b_j^{(n)}$

$$\tilde{b}_j^{(n)} = b_j^{(n)}(1 + \phi_j^{(n)}), \tag{28}$$

where

$$|\phi_j^{(n)}| \leq \epsilon_M(n+1)\begin{cases} 2 & \text{(real case)} \\ 5+\sqrt{2} & \text{(complex case)} \end{cases} + \mathcal{O}(\epsilon_M^2) \tag{29}$$

and for $\tilde{B}_n$, $n = 1, 2 \ldots, N$

$$\tilde{B}_n = \sum_{j=0}^{n} \tilde{b}_j^{(n)}(1 + \psi_j^{(n)}),$$

where

$$|\psi_j^{(n)}| \leq \epsilon_M(n+1)\begin{cases} 3 & \text{(real case)} \\ 6+\sqrt{2} & \text{(complex case)} \end{cases} + \mathcal{O}(\epsilon_M^2).$$

Finally, this and (28), (29) lead to (27), which proves the theorem. □

Remark 2: *In the case of the divided differences algorithm (formula (3)) the constants are smaller, namely (cf. [13])*

$$|\delta_j^{(n)}| \leq \epsilon_M(n+1)\begin{cases} 3 & \textit{(real case)} \\ 6+\sqrt{2} & \textit{(complex case)} \end{cases} + \mathcal{O}(\epsilon_M^2).$$

Acknowledgements

We are grateful to anonymous referees for their various comments and suggestions, especially for bringing our attention to the result in Remark 1.

References

[1] Baglama, J., Calvetti, D., Reichel, L.: Fast Leja points. ETNA *7*, 124–140 (1998).
[2] Björck, Å., Pereyra, V.: Solution of Vandermonde systems of equations. Math. Comput. *24*(112), 893–903 (1970).
[3] Brent, R. P.: Stability of fast algorithms for structured linear systems. Technical Report TR-CS-97-18, ANU, September 1997.
[4] Calvetti, D., Reichel, L.: On the evaluation of polynomial coefficients. Numer. Algorith. *33*, 153–161 (2003).
[5] Dahlquist, G., Björck, Å.: Numerical methods. Englewood Cliffs, NJ: Prentice-Hall 1974.
[6] Higham, N. J.: Accuracy and stability of numerical algorithms. Philadelphia: SIAM 1996.
[7] Jankowska, J., Jankowski, M.: Przegląd metod i algorytmów numerycznych, cz. 1. Warszawa: WNT 1981 (in Polish).
[8] Kahan, W., Farkas, I.: Algorithm 167-calculation of confluent divided differences. Comm. ACM *6*, 164–165 (1963).
[9] Leja, F.: Sur certaines suits liées aux ensemble plan et leur application à la representation conforme. Ann. Polon. Math. *4*, 8–13 (1957).
[10] McCurdy, A., Ng, K. C., Parlett, B. N.: Accurate computation of divided differences of the exponential function. Math. Comput. *43*(168), 501–528 (1984).

[11] Rack, H. J., Reimer, M.: The numerical stability of evaluation schemes for polynomials based on the Lagrange interpolation form. BIT *22*, 101–107 (1982).
[12] Reichel, L.: Newton interpolation at Leja points. BIT *30*, 332–346 (1990).
[13] Smoktunowicz, A.: Stability issues for special algebraic problems. Ph.D. thesis, University of Warsaw, 1981 (in Polish).
[14] Smoktunowicz, A.: Backward stability of Clenshaw's algorithm. BIT *42*(3), 600–610 (2002).
[15] Smoktunowicz, A., Kosowski, P., Wróbel, I.: How to overcome the numerical instability of the scheme of divided differences? arXiv.org/pdf/math.NA/0407195, 2004.
[16] Smoktunowicz, A., Wróbel, I.: On improving the accuracy of Horner's and Goertzel's algorithms. Numer. Algorith. *38*, 243–258 (2005).
[17] Tal-Ezer, H.: High degree polynomial interpolation in Newton form. SIAM J. Sci. Stat. Comput. *12*, 648–667 (1991).
[18] Werner, W.: Polynomial interpolation: Lagrange versus Newton. Math. Comput. *43*(167), 205–217 (1984).
[19] Wilkinson, J. H.: Rounding errors in algebraic processes. Notes on Applied Science No. 32. London: Her Majesty's Stationary Office 1963.

A. Smoktunowicz, I. Wróbel and P. Kosowski
Faculty of Mathematics and Information Science
Warsaw University of Technology
Pl. Politechniki 1
00-661 Warsaw
Poland
e-mails: smok@mini.pw.edu.pl; i.wrobel@mini.pw.edu.pl