Appl. Math. Lett. Vol. 2, No. 3, pp. 255–258, 1989
Printed in Great Britain

0893-9659/89 $3.00 + 0.00
Maxwell Pergamon Macmillan plc

Fast Evaluation and Interpolation at the Chebyshev Sets of Points

VICTOR PAN

State University of New York at Albany

Abstract. Stable polynomial evaluation and interpolation at n Chebyshev or adjusted (expanded) Chebyshev points is performed using $O(n \log^2 n)$ arithmetic operations, to be compared with customary algorithms either using on the order of n^2 operations or being unstable. We also evaluate a polynomial of degree d at the sets of n Chebyshev or adjusted (expanded) Chebyshev points using $O(d \log d \log n)$ if $n \leq d$ or $O((d \log d + n) \log d)$ arithmetic operations if $n > d$.

1. INTRODUCTION

Consider the set of Fourier points on the unit circle in the complex plane,

$$\{\omega^{2k+1}, \quad k = 0, 1, \ldots, n-1\}, \tag{1}$$

$\omega = \exp\left(\frac{\pi\sqrt{-1}}{2n}\right)$ being a primitive $(4n)$-th root of 1, and also the Chebyshev set of the real coordinates of these points,

$$\{x_k = \cos\left(\frac{2k+1}{2n}\pi\right), \quad k = 0, 1, \ldots, n-1\} \tag{2}$$

([4], [5], [7]). Fast Fourier transform (FFT) is a very effective means of interpolation to a function at the Fourier set (1) by a polynomial and of the evaluation of a polynomial at such a set of points. We will present fast stable algorithms for the evaluation and interpolation at the Chebyshev set (2), which are almost as efficient as fast Fourier transform (FFT) at the set (1), provided that $n+1 = 2^h$ is an integer power of 2; those computations at the Chebyshev set (2) are highly important for the approximation to functions by polynomials and for stable polynomial evaluation (see Section 4 for our comments on the computational cost and for a discussion). The results apply also to the sets of adjusted (expanded) Chebyshev points $\{y_k = ax_k + b, \quad k = 0, 1, \ldots, n-1\}$ for two real constants a and b.

Hereafter all logarithms are to the base 2.

2. EVALUATION

Next we will extend the known scheme for FFT to polynomial evaluation at the set (2). Given a polynomial $P(x) = \sum_{i=0}^{d} P_i x^i$, we write $P(x) = P_0(x^2) + xP_1(x^2)$, $P_0(x^2) = \sum_j P_{2j} x^{2j}$, $P_1(x^2) = \sum_j P_{2j+1} x^{2j}$, $Q_h(y) = P_h\left(\frac{1-y}{2}\right)$ for j ranging from 0 to $\lfloor \frac{d}{2} \rfloor$, $h = 0, 1$, $y = 1 - 2x^2$, so that

$$P(x) = Q_0(y) + xQ_1(y), \quad Q_h(y) = P_h(x^2), \quad y = 1 - 2x^2, \quad h = 0, 1. \tag{3}$$

Those equations reduce the evaluation of $P(x)$ at the n point x set (2) to the evaluation of two smaller degree (at most half-degree) polynomials $Q_0(y)$, $Q_1(y)$ at the Chebyshev $\frac{n}{2}$ points y-set

This research has been supported by NSF Grant CCR-8805782

$$y_k = \cos\left(\frac{2k+1}{n}\pi\right) \quad \text{for } k = 0, 1, \ldots, \frac{n-2}{2}, \tag{4}$$

for $1 - 2\cos^2\alpha = \cos(2\alpha)$.

Let $E(d,n)$ denote the cost of evaluation of $P(x)$ at the set (2), for even d and n. Then the above reduction shows that

$$E(d,n) \le 2E\left(\frac{d}{2}, \frac{n}{2}\right) + Overhead(d,n)$$

where Overhead(d,n) denotes the overall cost a) of computing the coefficients of the polynomials $Q_h(y)$, given the coefficients of $P_h(x)$ for $h = 0, 1$, and b) of computing $P(x)$, given $P_0(x^2)$ and $P_1(x^2)$ for x ranging over the set (2). Stage a) can be reduced to polynomial multiplication ([2]), and thus to three FFTs at $\le d+1$ points, that is, to $O(d\log d)$ arithmetic operations; the cost of Stage b) is $2n+2$ arithmetic operations. Let n be an integer power of 2. We will apply such reduction recursively, until the degree of polynomials or the cardinality of the Chebyshev set decrease to 1, whichever comes first, and finally compute $P(x)$ at the set (2) using $O((d\log d + n)\log d)$ (if $d \le n$) or $O(d\log d\log n)$ (if $n \le d$) arithmetic operations. If $n = O(d)$, the overall asymptotic cost is dominated by the cost of the shifts of the variable by 1 (Stage a)); thus the computation will be faster where such shifts can be performed faster.

3. Interpolation

The known fast $O(n\log^2 n)$ time algorithm for interpolation ([1],[3]) is not stable only at the auxilliary stage of computing the derivative $L'(x_i)$ for $i = 0, 1, \ldots, n-1$ where $L(x) = \prod_i (x - x_i)$. If, however, $x_0, \ldots, x_{n-1}$ are Chebyshev's points and n is a power of 2, that stage can be made both stable and fast (see the previous section). Alternatively, we may directly apply the converse version of the evaluation algorithm of the previous section. Indeed, the substitution $y = 1 - 2x^2$ transforms two distinct points x_i and x_j of the set (2) into each point y_k of the set (4); then (3) implies that $P(x_h) = Q_0(y_k) + x_h Q_1(y_k)$ for $h = i$ and $h = j$. This reduces the interpolation problem of finding a polynomial $P(x)$ of degree d given its values at the set (2) to finding two polynomials $Q_0(y)$ and $Q_1(y)$ of degree $\le \frac{d}{2}$, given their values at the half-size set (4). Applying that reduction recursively yields the cost bound $O(n\log^2 n)$ for interpolation provided that n is an integer power of 2.

Next, we will relax the latter assumption and will present an alternative algorithm, also fast and stable, that uses $O(n\log n)$ operations (for any integer n) in order to recover a distinct representation (see (6) below) of the polynomial $P(x)$ from its values at the set (2). We will use the functions

$$x = \frac{z^2+1}{2z}, \quad z = x \pm \sqrt{x^2 - 1} \tag{5}$$

which map the sets (1) and (2) into each other. Substitute (5) and rewrite the polynomial $P(x) = \sum_{i=0}^{d} P_i x^i$ as follows:

$$\begin{aligned} P(x) = p(z) = p(\tfrac{1}{z}) &= \frac{r(z)}{z^d}, \\ p(z) = \sum_{i=0}^{d} p_i z^i, \quad p_d &= \frac{P_d}{2^d}, \\ r(z) = z^d \sum_{i=0}^{d} p_i (z^i + z^{-i}), & \end{aligned} \tag{6}$$

Let us assume that $d = n - 1$. Due to the equation (6), $2n$ multiplications suffice in order to compute $r(z)$ and $r(\frac{1}{z})$ at the Fourier set (1) given the values of $P(x)$ at the Chebyshev

set (2). This gives the values of the polynomial $s(z) = r(\omega z)$ for z ranging over the set $\exp(\frac{j\pi\sqrt{-1}}{2n})$, $j = 0, 1, \ldots, 2n-1$. Then we may apply FFT at $2n$ points and compute the coefficients of the polynomial $s(z) = r(\omega z)$. After that we may immediately recover the coefficients p_i of $r(z)$ and of $p(z)$. If we only know that $d < n$, we may apply the above algorithm to both $P(x)$ and $P(x) + x^n$. In some cases the representation by $p_0, \ldots, p_d$ may replace the coefficient-wise representation of the polynomial $P(x)$. In particular, if we want to compute $P(x)$ at a point x not lying in the set (2), we may compute $z = x + \sqrt{x^2-1}$, $\frac{1}{z} = x - \sqrt{x^2-1}$, and then $P(x) = p(z) + p(\frac{1}{z})$. For a point z this will cost about $4d$ arithmetic operations, excluding the cost of the evaluation of the coefficients p_i of $r(z)$. Note that the coefficients p_i are real if all the P_i are real and that z is real if $x \geq 1$.

4. Discussion and Open Problems

Interpolation at the Chebyshev sets of points, as well as at the Chebyshev adjusted sets, is a well-known means of approximation (see [4],[5],[7]). The evaluation of a function at the Chebyshev sets can be replaced by the evaluation of a high degree approximating polynomial at that set, which then can be replaced by the lower degree interpolation polynomial. The evaluation of a polynomial at the Chebyshev points can be used in order to represent that polynomial in Chebyshev's basis, which then can be used for the stable evaluation of that polynomial ([6], p. 249).

The customary methods for the interpolation at Chebyshev points do not exploit their specifics and require on the order of d^2 arithmetic operations (see [4],[9]). Thus our improvement to $O(d\log^2 d)$ is substantial. There are general interpolation algorithms also running in $O(d\log^2 d)$ time ([1],[3]), but they recursively use polynomial division, which creates stability problems. This is not the case with our algorithms for we use FFT rather than polynomial divisions. The same conclusions apply to our evaluation methods. The recent ingenious algorithm of [8] may compete with ours but only for very rough approximate evaluation of polynomials, for the time cost of Rokhlin's algorithm is on the order of $d\log(\frac{1}{\epsilon}) + n\log^3(\frac{1}{\epsilon})$ where ϵ is the output error bound (and that algorithm does not apply to interpolation). Now, we state our final question: besides the sets (1), (2) and similar ones, what are other interesting real and complex sets where polynomial interpolation and evaluation are particularly simple? A natural approach is to start with the sets that can be easily transformed into the sets (1) and/or (2). Another possibility is to extend the idea used in Sections 2 and 3 to the evaluation ind interpolation at the sets whose cardinality is recursively decreased by 50% via the substitution of the variable of the form $y = a + bx^2$ for two constants a and b. The sets (1) and (2) satisfy that property.

REFERENCES

1. A.V. Aho, J.E. Hopcroft, J.D. Ullman, "The Design and Anlalysis of Computer Algorithms," Addison-Wesley, 1976.
2. A.V. Aho, K. Steiglitz, J.D. Ullman, *Evaluating Polynomials at Fixed Set of Points*, SIAM J. on Computing **4** (1975), 533–539.
3. A. Borodin, I. Munro, *The Computational Complexity of Algebraic and Numeric Problems*, American Elsevier, New York (1975).
4. C.D. Conte, C. de Boor, "Elementary Numerical Analysis: an Algorithmic Approach," McGraw- Hill, N.Y., 1980.
5. G. Dahlquist, A. Bjoŕk, "Numerical Methods," Prentice-Hall, New Jersey, 1974.
6. P. Henrici, "Essentials of Numerical Analysis with Pocket Calculator Demonstrations," Wiley, New York, 1982.
7. M.S. Henry, *Approximation by Polynomials: Interpolation and Optimal Nodes*, Math. Monthly **91**, 8, (1984), 497–499.
8. V. Rokhlin, *A Fast Algorithm for the Discrete Laplace Transformation* , Research Report YaleU/DCS/RR-509,*Dept. of Computer Science, Yale Univ.* (1987).
9. W. Werner, *Polynomial Interpolation: Lagrange versus Newton*, Math. of Computations **43** 167 (1984), 205–217.

State University of New York at Albany,
Computer Science Department; Albany, New York 12222