An algorithm is presented for the rapid evaluation of expressions of the form

$$\sum_{j=1}^{m} \alpha_j \cdot e^{-\beta_j \cdot x}$$

at multiple points $x_1, x_2, \cdots, x_n$. In order to evaluate the above sum at n points, the algorithm requires order $O(n+m)$ operations, and a simple modification of the scheme provides an order $O(n)$ procedure for the evaluation of an order n polynomial at n arbitrary real points. The algorithm is numerically stable, and its practical usefulness is demonstrated by numerical examples.

A Fast Algorithm for the Discrete Laplace Transformation

V. Rokhlin

Research Report YALEU/DCS/RR-509
January 1987

The author was supported in part by the Office of Naval Research under Grant N00014-86-K-0310

1. Introduction

In this paper, we present an algorithm for the rapid evaluation of expressions of the form

$$\sum_{j=1}^{m} \alpha_j \cdot e^{-\beta_j \cdot x}, \tag{1}$$

where $x \geq 0$, $\tilde{\alpha} = \{\alpha_1, \alpha_2, \cdots, \alpha_m\}, \tilde{\beta} = \{\beta_1, \beta_2, \cdots, \beta_m\}$ are two finite sequences of real numbers and $\beta_j \geq 0$ for all $1 \leq j \leq m$. To evaluate the sum (1) at n arbitrary points on the real axis, the algorithm requires a number of arithmetic operations proportional to

$$(n + m \cdot log_2(\frac{1}{\epsilon})) \cdot (log_2(\frac{1}{\epsilon}))^2, \tag{2}$$

where ϵ is the precision with which the calculations are being performed, and in most cases likely to be encountered in practice, the estimate (2) can be reduced to

$$(n + m) \cdot log_2(\frac{1}{\epsilon}) \tag{3}$$

(see Observations 7.1, 7.2 below).

The evaluation of expressions of the form (1) is closely related to several classical problems in the theory of computation. For example, the problem of rapidly evaluating a polynomial

$$P(t) = \sum_{j=1}^{m} P_j \cdot t^j \tag{4}$$

at m different points is readily reduced to the form (1) by the obvious substitution $x = log(t)$. The classical algorithm for evaluating (4) at m points has an asymptotic complexity $O(m \cdot log^2(m))$ (see, for example, [1,2]), making (3) a moderate improvement over previously available results, so far as the asymptotic CPU time estimate is concerned. On the other hand, the algorithm of the present paper is numerically stable, and our numerical experiments (see Section 8) indicate that in practical calculations, it is extremely efficient, making it a method of choice whenever expressions of the form (1) have to be evaluated at large numbers of points.

Remark 1.1. Classical algorithms for the rapid manipulation of polynomials are purely algebraic, and are applicable to polynomials over a wide class of fields. On the other hand, the algorithm presented here is based on approximation theory (i.e it relies on certain facts from real analysis) and is restricted to polynomials over the field of real numbers. While it can be generalized to certain other fields, detailed investigation of such generalizations is outside the scope of this paper, and will be reported at a later date.

2. Relevant Facts From Approximation Theory

Suppose that a, b are a pair of real numbers such that $a < b$, and that $k \geq 2$ is an integer. Chebychev nodes $t_1, t_2, \cdots, t_k$ on the interval $[a, b]$ are defined by the formula

$$t_i = \frac{a+b}{2} + \frac{a-b}{2} \cdot cos(\frac{2\,i+1}{k} \cdot \frac{\pi}{2}). \tag{5}$$

For a function $f : [a,b] \to R^1$, we will denote by $P^k_{a,b,f}$ the order $k-1$ Chebychev approximation to the function f on the interval $[a,b]$, i.e the (unique) polynomial of order $k-1$ such that $P^k_{a,b,f}(t_i) = f(t_i)$ for all $i = 1, 2, \cdots, k$. There exist several expressions for the polynomial $P^k_{a,b,f}$, and the one we will use in this paper is

$$P^k_{a,b,f}(t) = \sum_{j=1}^{k} u_j(t) \cdot f(t_j) \tag{6}$$

with

$$u_j(t) = \frac{\prod_{i=1,i\neq j}^{k}(t-t_i)}{\prod_{i=1,i\neq j}^{k}(t_j-t_i)}. \tag{7}$$

The following well-known lemma provides an error estimate for Chebychev approximations. It is the principal analytical tool of this paper, and can be found, in a somewhat different form, in [3].

Lemma 2.1. If $f \in c^k[a,b]$ (i.e. f has k continous derivatives on the interval $[a,b]$), then for any $t \in [a,b]$,

$$\mid P^k_{a,b,f}(t) - f(t) \mid \leq \frac{M}{k!} \cdot \frac{(b-a)^k}{4^k} \tag{8}$$

with

$$M = \max_{t\in[a,b]} \mid f^{(k)}(t) \mid . \tag{9}$$

Furthermore, for any $k \geq 2$ and $t \in [a,b]$,

$$\sum_{j=1}^{k} u_j^2(t) \leq 2, \tag{10}$$

and

$$\sum_{j=1}^{k} \mid u_j(t) \mid \leq 2 + \frac{2}{\pi} \cdot log(k). \tag{11}$$

In the present paper, the above lemma will be used in the special case where $0 < a < b$, and $f(t) = e^{-\gamma \cdot t}$, with $\gamma > 0$. Under these conditions, the expression (8) assumes the form

$$\mid P^k_{a,b,f}(t) - f(t) \mid \leq \frac{\gamma^k}{k!} \cdot \frac{(b-a)^k}{4^k} \cdot e^{-\gamma \cdot t}, \tag{12}$$

and the following lemma provides a form of the estimate (12) independent of γ.

Lemma 2.2. If under the conditions of Lemma 2.1, $f(t) = e^{-\gamma \cdot t}, b = 2a$, and $a > 0$, then

$$\mid P^k_{a,b,f}(t) - f(t) \mid \leq \frac{1}{4^k} \tag{13}$$

for all $k \geq 2$ and $t \in [a, b]$.

Proof. Obvously, for $t \in [a, 2a]$, the estimate (12) can be rewritten in the form

$$| P^k_{a,b,f}(t) - f(t) | \leq \frac{\gamma^k}{k!} \cdot \frac{a^k}{4^k} \cdot e^{-\gamma \cdot a}. \tag{14}$$

Differentiating the latter expression with respect to γ, we find that its maximum is achieved at

$$\gamma = \frac{k}{a}. \tag{15}$$

Now, substituting (15) into (14) and using Stirling's formula, we obtain

$$| P^k_{a,b,f}(t) - f(t) | \leq \frac{1}{k!} \cdot (\frac{k}{a})^k \cdot \frac{a^k}{4^k} \cdot e^{-\frac{k}{a} \cdot a} \leq \frac{1}{4^k} \cdot \frac{k^k}{k!} \cdot e^{-k} \leq \frac{1}{4^k}. \tag{16}$$

3. Exact Statement of the Problem

In the description of the algorithm below, we will assume that:

a) $\tilde{\alpha} = \{\alpha_1, \alpha_2, \cdots, \alpha_m\}, \tilde{\beta} = \{\beta_1, \beta_2, \cdots, \beta_m\}, \tilde{x} = \{x_1, x_2, \cdots, x_n\}$ are three finite sequences of real numbers.
b) The sequences $\tilde{\beta}$ and $\tilde{x}$ are monotonically increasing.
c) $\beta_1 \geq 0$.
d) $x_1 \geq 0$.
e) We would like to evaluate the sums

$$S_{\alpha,\beta}(x_k) = \sum_{j=1}^{m} \alpha_j \cdot e^{-\beta_j \cdot x_k} \tag{17}$$

for all $k = 1, 2, \cdots, n$ with a relative accuracy $\epsilon > 0$, i.e. we would like to find a number $\tilde{S}_{\alpha,\beta}(x_k)$ such that

$$\frac{| \tilde{S}_{\alpha,\beta}(x_k) - S_{\alpha,\beta}(x_k) |}{\sum_{j=1}^{m} | \alpha_j |} \leq \epsilon \tag{18}$$

for each $k \in [1, n]$.

Remark 3.1. As has been mentioned in the Introduction, the problem of evaluating a polynomial of order m at n points is easily reduced to the form (17). Indeed, suppose that a polynomial of the form (4) has to be evaluated at a monotonically increasing finite sequence of points $t_1, t_2, \cdots, t_n$. It can be assumed without a loss of generality that $0 \leq t_k \leq 1$ for all $k = 1, 2, \cdots, n$, and we will introduce a new variable $x = -log(t)$, and denote $-log(t_k)$ by x_k. Thus, evaluation of the polynomial (4) at a monotonically increasing finite sequence of points has been reduced to evaluating the expression

$$\sum_{j=1}^{m} P_j \cdot e^{-j \cdot t} \tag{19}$$

at a monotonically decreasing finite sequence of points $\tilde{x} = \{x_1, x_2, \cdots, x_n\}$. Finally, by reversing the order of the sequence $\tilde{x}$, we reduce the evaluation of the polynomial (4) at the points $t_1, t_2, \cdots, t_n$ to the standard problem formulated above.

4. Notation

In this section, we will introduce several definitions to be used in the description of the algorithm in Sections 5, 6 below. Throughout this section, we will assume that we are dealing with the problem described in Section 3, and that q is an integer whose particular value is to be determined later.

We will denote by M the smallest integer number such that

$$\frac{\beta_m \cdot x_n}{2^{M-1}} < \epsilon. \tag{20}$$

We will define a finite sequence $\{U_i\}, i = 1, 2, \cdots, M$ of intervals on the real axis by the formulae

$$\begin{aligned} U_i &= [\frac{\beta_m}{2^i}, \frac{\beta_m}{2^{i-1}}] \ for \ 1 \le i \le M-1, \\ U_M &= [0, \frac{\beta_m}{2^{M-1}}]. \end{aligned} \tag{21}$$

Similarly, we will define a finite sequence $\{V_i\}, i = 1, 2, \cdots, M$ of intervals by the formulae

$$\begin{aligned} V_i &= [\frac{x_n}{2^i}, \frac{x_n}{2^{i-1}}] \ for \ 1 \le i \le M-1, \\ V_M &= [0, \frac{x_n}{2^{M-1}}]. \end{aligned} \tag{22}$$

For any $i = 1, 2, \cdots, M$, we will denote by $\tilde{\beta}_i$ the subset of β consisting of all points β_l such that $\beta_l \in U_i$, and for any $i = 1, 2, \cdots, M$, we will denote by $\tilde{x}_j$ the subset of x consisting of all points x_l such that $x_j \in V_i$.

For each $i = 1, 2, \cdots, M, m_i$ will denote the number of elements in $\tilde{\beta}_i$. Similarly, for each $i = 1, 2, \cdots, M, n_i$ will denote the number of elements in $\tilde{x}_i$.

Remark 4.1. Obviously, depending on the distributions of the points β_i and x_i, the M can be fairly large. However, the total number $\overline{M}$ of such i that $m_i \neq 0$ is bounded by m, and the total number $\overline{N}$ of such i that $n_i \neq 0$ is bounded by n. For obvious reasons, we will refer as empty to intervals U_i, V_j such that $m_i = 0$ and $n_j = 0$. In the opposite case, the intervals will be referred to as non-empty.

For each $i = 1, 2, \cdots, M$, and $j = 1, 2, \cdots, q$, we will denote by β^i_j the j-th Chebychev node on the interval U_i.

Similarly, for each $i = 1, 2, \cdots, M$, and $j = 1, 2, \cdots, q$, we will denote by x^i_j the j-th Chebychev node on the interval V_i.

For each $k = 1, 2, \cdots, M$, and i such that $\beta_i \in U_k$, we will define the finite sequence $\{u^k_{i,j}\}, j = 1, 2, \cdots, q$ by the formula

$$u_{i,j}^k = \prod_{l=1, l \neq j}^{q} \frac{\beta_i - \beta_l^k}{\beta_j^k - \beta_l^k}. \tag{23}$$

For each $k = 1, 2, \cdots, M$, and $i = 1, 2, \cdots, q$, we will define a real number u_i^k by the formula

$$u_i^k = \sum_{\beta_j \in U_k} \alpha_j \cdot u_{i,j}^k. \tag{24}$$

Observation 4.1. Due to Lemma 2.2, the expression

$$\phi_i^k(t) = \sum_{j=1}^{q} u_{i,j}^k \cdot e^{-\beta_j^k \cdot t} \tag{25}$$

can be viewed as an approximation to the function $e^{-\beta_i \cdot t}$. Furthermore, for any $t \in [0, \infty]$,

$$\mid \phi_i^k(t) - e^{-\beta_i \cdot t} \mid \leq \frac{1}{4^q}. \tag{26}$$

Combining (24), (25), (26) with the triangle inequality, we easily see that the sum

$$\begin{aligned} \phi_k(t) &= \sum_{i=1}^{q} u_i^k \cdot e^{-\beta_i^k \cdot t} = \sum_{i=1}^{q} \sum_{\beta_j \in U_k} \alpha_j \cdot u_{j,i}^k \cdot e^{-\beta_i^k \cdot t} \\ &= \sum_{\beta_j \in U_k} \alpha_j \cdot \sum_{i=1}^{q} u_{j,i}^k \cdot e^{-\beta_i^k \cdot t} \end{aligned} \tag{27}$$

can be viewed as an approximation to

$$\sum_{\beta_j \in U_k} \alpha_j \cdot e^{-\beta_j \cdot t}, \tag{28}$$

and that

$$\mid \phi_k(t) - \sum_{\beta_j \in U_k} \alpha_j \cdot e^{-\beta_j \cdot t} \mid \leq \frac{1}{4^q} \cdot \sum_{\beta_j \in U_k} \mid \alpha_j \mid. \tag{29}$$

Furthermore, combining (11) with (24) and using the triangle inequality, we obtain

$$\sum_{i=1}^{q} \mid u_i^k \mid \leq \sum_{j \in U_k} (\mid \alpha_j \mid \cdot \sum_{i=1}^{q} \mid u_{i,j}^k \mid) \leq (2 + \frac{2}{\pi} \cdot log(q)) \cdot \sum_{\beta_j \in U_k} \mid \alpha_j \mid. \tag{30}$$

Given $k = 1, 2, \cdots, M$, and $i = 1, 2, \cdots, q$, we will define a real number f_i^k by the expression

$$f_i^k = \sum_{j=\nu_k+1}^{\mu_k - 1} \sum_{l=1}^{q} u_l^j \cdot e^{-\beta_l^j \cdot x_i^k}. \tag{31}$$

For each $k = 1, 2, \cdots, M$, and $1 \leq j \leq n$ such that $x_j \in V_k$, we will define f_j by the formula

$$f_j = \sum_{l=1}^{q} v_{l,j}^k \cdot f_l^k, \tag{32}$$

with the coefficients $v_{l,j}^k$ defined by the formula

$$v_{i,j}^k = \prod_{i=1, i \neq l}^{q} \frac{x_j - x_l^k}{x_i^k - x_l^k}. \tag{33}$$

Observation 4.2. Due to Lemma 2.2, for any $j = 1, 2, \cdots, n$ and k such that $x_j \in V_k$, f_j can be viewed as an approximation to the expression

$$\tilde{f}_j = \sum_{i=\nu_k+1}^{\mu_k - 1} \sum_{l=1}^{q} u_l^i \cdot e^{-\beta_l^i \cdot x_j}, \tag{34}$$

and

$$| f_j - \tilde{f}_j | \leq \frac{1}{4^q} \cdot \sum_{i=\nu_k+1}^{\mu_k - 1} \sum_{l=1}^{q} | u_l^i | . \tag{35}$$

Combining (35), (29) (30), and using the triangle inequality, we conclude that

$$| f_j - \sum_{i=1}^{m} \alpha_i \cdot e^{-\beta_i \cdot x_j} | \leq \frac{1}{4^q} \cdot (3 + \frac{2}{\pi} \cdot log(q)) \cdot \sum_{i=1}^{m} | \alpha_i | . \tag{36}$$

for any $j = 1, 2, \cdots, n$. Now, for any given ϵ and $q > 2 \cdot log_4(\epsilon)$,

$$| f_j - \sum_{i=\nu_k+1}^{\mu_k - 1} \alpha_i \cdot e^{-\beta_i \cdot x_j} | \leq \epsilon \cdot \sum_{i=1}^{m} | \alpha_i | . \tag{37}$$

For any $i = 1, 2, \cdots, M - 1$, we will denote by ν_i the largest integer such that

$$\nu_i < log_2(\beta_m \cdot x_n) - i - log_2(log_e(\frac{1}{\epsilon})). \tag{38}$$

Similarly, for any $i = 1, 2, \cdots, M - 1$, we will denote by μ_i the smallest interger such that

$$\mu_i > log_2(\beta_m \cdot x_n) - i - log_2(\frac{1}{\epsilon}). \tag{39}$$

For any $k = 1, 2, \cdots, M$, we will define the subset W_k of the interval $[0, \beta_n]$ by the formula

$$W_k = \bigcup_{i \geq \mu_k} V_i, \tag{40}$$

and denote by S_k the sum

$$S_k = \sum_{\beta_j \in W_k} \alpha_j. \tag{41}$$

Observation 4.3. It is easy to see that if $x \in U_i$ and $\beta \in V_j$ with $j \leq \nu_i$, then

$$e^{-x \cdot \beta} \leq \epsilon. \tag{42}$$

Similarly, if $x \in U_i$ and $\beta \in V_j$ with $j \geq \mu_i$, then

$$| e^{-x \cdot \beta} - 1 | \leq \epsilon. \tag{43}$$

Fuerthermore, for any $i = 1, 2, \cdots, M-1$,

$$\mu_i - \nu_i \leq 2 \cdot log_2(\frac{1}{\epsilon}). \tag{44}$$

In other words, given $x \in U_i$ and $\beta \in V_j$, one of three possible situations obtains:

a) $j \leq \nu_i$. In this case, $e^{-\beta \cdot x}$ can be approximated by 0 with a precision ϵ.

b) $j \geq \mu_i$. In this case, $e^{-\beta \cdot x}$ can be approximated by 1 with a precision ϵ.

c) $\nu_i \leq j \leq \mu_i$. In this case, $e^{-\beta \cdot x}$ can not be approximated by either 0 or 1. However, the total number of indices j for which this situation obtains is bounded by $2 \cdot log_2(\frac{1}{\epsilon})$, independently of $\tilde{x}, \tilde{\beta}$, or i.

5. Informal Description of the Algorithm.

We will illustrate the idea of the algorithm on a simplified example. Namely, we will assume that $\beta_i \in U_1$, i.e.

$$\frac{\beta_m}{2} \leq \beta_i \leq \beta_m \tag{45}$$

for all $i = 1, 2, \cdots, m$, and $x_j \in V_1$, i.e.

$$\frac{x_n}{2} \leq x_j \leq x_n \tag{46}$$

for all $j = 1, 2, \cdots, n$.

Consider the function $e^{-\beta \cdot x}$ with $\beta \in U_1, x \in V_1$. Fixing x and viewing $e^{-\beta \cdot x}$ as a function of β, we construct its q-point Chebychev approximation $\psi_x^q(\beta)$ on the interval U_1. Due to (6),

$$\psi_x^q(\beta) = \sum_{j=1}^{q} u_j(\beta) \cdot e^{-\beta_j^1 \cdot x}, \tag{47}$$

with the functions u_j defined by (7), and the coefficients β_j^i defined in Section 4. According to Lemma 2.2,

$$| \psi_x^q(\beta) - e^{-\beta \cdot x} | \leq \frac{1}{4^q}, \tag{48}$$

and, given a fixed precision ϵ, we can choose $q \sim 2 \cdot log_4(\frac{1}{\epsilon})$, and in all subsequent calculations replace $e^{-\beta \cdot x}$ with $\psi_x^q(\beta)$. Combining (48) with the triangle inequality, we obtain the estimate

$$\mid \sum_{j=1}^{m} \alpha_j \cdot \psi_x^q(\beta_j) - \sum_{j=1}^{m} \alpha_j \cdot e^{-\beta \cdot x} \mid \leq \frac{1}{4^q} \cdot \sum_{j=1}^{m} \mid \alpha_j \mid \tag{49}$$

for any $x \in [0, +\infty]$, and due to (45), the latter can be rewritten in the form

$$\mid \sum_{i=1}^{q} \psi_i \cdot e^{-x_k \cdot \beta_i^1} - \sum_{j=1}^{m} \alpha_j \cdot e^{-\beta \cdot x} \mid \leq \frac{1}{4^q} \cdot \sum_{j=1}^{m} \mid \alpha_j \mid, \tag{50}$$

with the coefficients $\psi_1, \psi_2, \cdots, \psi_q$ defined by the formula

$$\psi_i = \sum_{j=1}^{m} \alpha_j \cdot u_i(\beta_j). \tag{51}$$

Now, instead of evaluating (17) at each of the points x_i, we start with evaluating the coefficients $\psi_i, i = 1, 2, \cdots, q$, which is, obviously, an order $O(m \cdot q)$ procedure. After that, we evaluate the expression

$$\sum_{i=1}^{q} \psi_i \cdot e^{-x_k \cdot \beta_i^1} \tag{52}$$

for all $k = 1, 2, \cdots, n$, which is an order $O(n \cdot q)$ procedure (evaluating a q-term expansion at n points). Thus, the total operation count becomes $O((n+m) \cdot q)$. Due to (49), in order to obtain a relative accuracy ϵ, q has to be of the order $log_4(\frac{1}{\epsilon})$, and we have reduced the computational complexity of evaluating (17) from $O(n \cdot m)$ to

$$O((n+m) \cdot log_4(\frac{1}{\epsilon})). \tag{53}$$

An alternative approach would be to calculate the coefficients ψ_i for $i = 1, 2, \cdots, q$ (order $m \cdot q$ operations), evaluate the expression

$$\sum_{i=1}^{q} \psi_i \cdot e^{-x_k^1 \cdot \beta_i^1} \tag{54}$$

for all $k = 1, 2, \cdots, q$ (order q^2 operations), and interpolate the expression (17) from the Chebychev nodes $x_1^1, x_2^1, \cdots, x_q^1$, to the points $x_1, x_2, \cdots, x_n$ (order $n \cdot q$ operations). The resulting CPU time estimate in this case is

$$O((n+m) \cdot q) + O(q^2) = O((n+m) \cdot log_4(\frac{1}{\epsilon}) + O((log_4(\frac{1}{\epsilon}))^2), \tag{55}$$

which is not substantially different from (53).

When the points $\beta_1, \beta_2, \cdots, \beta_m$ and $x_1, x_2, \cdots, x_n$ do not satisfy the inequalities (45), (46), the above approach can not be used in such straightforward manner. However, for any $i, j \in [1, M]$, Lemma 2.2 can be used separately on each of the intervals U_i, V_j, with the results combined to obtain an approximation to (17). This is done in the following section, resulting in an order

$$O((n+m) \cdot log(\frac{1}{\epsilon}) + O(n \cdot (log(\frac{1}{\epsilon}))^3) \qquad (56)$$

algorithm for evaluating (17) at n points with a relative precision ϵ.

6. Detailed Description of the Algorithm

Algorithm

Stage 1.

Comment [Choose parameters and perform geometrical preprocessing.]

Choose precision ϵ to be achieved. Set $q = 2 \cdot log(\frac{1}{\epsilon})$. Construct the intervals U_i, V_i, and the sets $\tilde{\beta}_i, \tilde{x}_i$ with $i = 1, 2, \cdots, M$.

Stage 2.

Comment [On each of the non-empty intervals U_k, evaluate the coefficients u_i^k in the expansions (27).]

Step 1.
Comment [Set all coefficients u_i^k to zero.]

do $k = 1, M-1, \tilde{\beta}_k \neq \emptyset$
  do $i = 1, q$
    set u_i^k to zero.
  end do
end do

Step 2.
Comment [For each β_i on each of the non-empty intervals U_k, evaluate $\alpha_i \cdot u_{j,i}^k$ and add it to the u_i^k.]

do $k = 1, M-1, \tilde{\beta}_k \neq \emptyset$
  do $i = 1, q$
    do $\beta_j \in U_k$
      Evaluate $u_{j,i}^k$ via formula (23) and add the product $\alpha_i \cdot u_{j,i}^k$ to u_i^k.
    end do
  end do
end do

Stage 3.

Comment [Evaluate f_i^k via formula (31) for all $k = 1, 2, \cdots, M$ such that $\tilde{x}_k \neq \emptyset$, and $i = 1, 2, \cdots, q$.]

do $k = 1, M-1, \tilde{x}_k \neq \emptyset$
  do $i = 1, q$
    evaluate the expression $f_i^k = \sum_{j=\nu_i+1}^{\mu_i-1} \sum_{l=1}^{q} u_l^j \cdot e^{-\beta_l^j \cdot x_i^k}$.
  end do
end do

Stage 4.

Comment [For each $j = 1, 2, \cdots, n$, evaluate f_j via formula (32).]

do $k = 1, M-1, \tilde{x}_k \neq \emptyset$
  do $x_j \in V_k$
    evaluate the expression $f_i = \sum_{l=1}^{q} v_{l,j}^k \cdot \phi_l^k$.
  end do
end do

Stage 5.

Comment [For each $k = 1, 2, \cdots, M$ and each $x_i \in V_k$, use Observation 4.3 to evaluate the sum $S_k = \sum_{\alpha_j \in W_k} \alpha_j$. Add the result to f_i, concluding the calculation.]

Step 1

Comment [Evaluate S_1.]

set $S_1 = \sum_{\beta_i \in U_1} \alpha_i$.

Step 2

Comment [Evaluate S_k recursively for $k = 2, 3, \cdots, M$.]

do $k = 2, M, \tilde{x}_k \neq \emptyset$
  evaluate S_k via the formula $S_k = S_{k-1} + \sum_{\beta_j \in U_k} \alpha_j$.
end do

Step 3

Comment [For all $k = 1, 2, \cdots, M$, and all i such that $x_i \in V_k$, add S_k to x_i, concluding the calculation.]

```
do k = 1, M, x̃_k ≠ ∅
    do x_i ∈ V_k
    add S_k to f_i.
    end do
end do
```

7. Complexity analysis

Stage number	Operation count	Explanation
Stage 1	$O(n+m)$	Each of the points $\beta_1, \beta_2, \cdots, \beta_m$ is assigned to a single interval U_i. Each of the points $x_1, x_2, \cdots, x_n$ is assigned to a single interval V_i.
Stage 2		
Step 1	$O(\overline{M} \cdot q)$	Each of the coefficients u_i^k, with $k = 1, 2, \cdots, M$, and $i = 1, 2, \cdots, q$ is set to zero.
Step 2	$O(m \cdot q^2)$	Each of the points $\beta_1, \beta_2, \cdots, \beta_m$ contributes to the coefficients $u_{j,i}^k$ with $j = 1, 2, \cdots, q$, and evaluating each of the coefficients $u_{j,i}^k$ requires order q work (see (23)) .
Stage 3	$O(\overline{M} \cdot q^2 \cdot (\frac{1}{\epsilon}))$	The sum (31) has to be evaluated at q nodes $x_1^k, x_2^k, \cdots x_q^k$ on each of non-empty intervals $V_1, V_2, \cdots, V_M$, and on the $k-th$ interval, it contains $\mu_k - \nu_k$ terms. However, due to (44)), $\mu_k - \nu_k \leq \epsilon$ for all $k = 1, 2, \cdots, M$.
Stage 4	$O(n \cdot q^2)$	The expression (32) has to be evaluated for each of the points $x_1, x_2, \cdots, x_n$, and evaluating each of the coefficients $v_{i,j}^k$ requires order q work (see (33)).
Stage 5		
Step 1	$O(m)$	The sum $S_1 = \sum_{\beta_i \in U_1} \alpha_i$ contains no more than m terms.
Step 2	$O(n+m)$	The total number of non-empty intervals V_i is bounded by n, and the total number of coefficients α_j is bounded by m.

Step 3 $O(n)$ Each of the numbers f_j is amended once.

Summing up the CPU times for all stages above, we obtain the following time estimate:

$$T_{total} = a \cdot m + b \cdot n + c \cdot m \cdot q^2 + d \cdot n \cdot q^2 + e \cdot \overline{M} \cdot q^2 \cdot log(\frac{1}{\epsilon}), \tag{57}$$

where the coefficients a, b, c, d, e depend on the computer system, language, implementation, etc. However, $\overline{M} \leq m$, and $q \sim log(\frac{1}{\epsilon})$, and the estimate (57) assumes the form

$$T_{total} = a \cdot m \cdot (\ log(\frac{1}{\epsilon}\ ))^3 + b \cdot n \cdot (\ log(\frac{1}{\epsilon}\ ))^2. \tag{58}$$

The estimate (58)) is independent of the locations of the points β_i, x_i in R^1, and does not depend on any precomputed data. The following two observations reduce it to

$$T_{total} = O((m+n) \cdot log(\frac{1}{\epsilon}) \tag{59}$$

for many problems of practical interest.

Observation 7.1. The term $b \cdot m \cdot q^2$ in (58) is associated with the Stage 3 of the algorithm and the grossly pessimistic estimate

$$\overline{M} \leq M \leq m. \tag{60}$$

According to (20),

$$M - 1 \leq log_2(\frac{\beta_m \cdot x_n}{\epsilon}) = log_2(\beta_m) + log_2(x_n) + log_2(\frac{1}{\epsilon}). \tag{61}$$

Normally, when calculations are performed on a physical computer, the exponential in the binary representation of a real number is bounded, and we will denote this bound by A. It immediately follows that in all cases,

$$\overline{M} \leq M \leq 3 \cdot log_2(A) + 1, \tag{62}$$

and the estimate (57) becomes

$$T_{total} = a \cdot m + b \cdot n + c \cdot m \cdot q^2 + d \cdot n \cdot q^2 + e \cdot log(A) \cdot q^2 \cdot log(\frac{1}{\epsilon}). \tag{63}$$

Observation 7.2. The terms $c \cdot m \cdot q^2$ and $d \cdot n \cdot q^2$ in (57) are associated with the Stages 2 and 4 of the algorithm, and with the fact that in order to evaluate each of the coefficients $u^k_{j,i}$ (or $v^k_{j,i}$), a $q-1$-term product of the form (24) (or (33)) has to be evaluated. Obviously, the coefficients $u^k_{j,i}$ depend only on the distribution of the points β_i, and not on that of x_i or α_i. Similarly, the coefficients $v^k_{j,i}$ depend only on the distribution of the points x_i, and not on that of β_i or α_i. Therefore, for fixed distributions of β_i and x_i, the coefficients $u^k_{j,i}$, $v^k_{j,i}$ can be precomputed and stored, reducing the total CPU time estimate to

$$T_{total} \sim a \cdot m \cdot q + b \cdot n \cdot q + +c \cdot log(A) \cdot q^2 \cdot log(\frac{1}{\epsilon}). \tag{64}$$

However, $q \sim log(\frac{1}{\epsilon})$, and $log(A)$ is fixed for given computer system and language. Thus, when $m, n \to \infty$,

$$T_{total} \sim (a \cdot m + b \cdot n) \cdot q \tag{65}$$

8. Numerical Results

A computer program has been written implementing the algorithm of this paper. The calculation is performed in two stages, each implemented by a separate subroutine. During the initialization stage, the coefficients $u^k_{i,j}, v^k_{i,j}$ are evaluated for given distributions of points $\beta_1, \beta_2, \cdots, \beta_m$, $x_1, x_2, \cdots, x_n$ (see Observation 7.2). During the second stage, the sums (17) are evaluated for a given set of weights $\alpha_1, \alpha_2, \cdots, \alpha_m$.

Remark 8.1. It is clear from Tables 1, 3, 5 that the first stage (initialization) tends to be several times more expensive than the second (evaluation). However, in most applications the algorithm has to be initialized once, with subsequent repeated evaluation of the sums (17) for varying sets of weights $\alpha_1, \alpha_2, \cdots, \alpha_m$. This situation is similar to that encountered for the Fast Fourier Transformation.

The program has been applied to a variety of situations, and three such examples are presented in this section, with the computations performed on a VAX-8600 computer. In each case, we performed the calculations in three ways: via the algorithm of the present paper in single precision arithmetic, directly in single precision arithmetic, and directly in double precision arithmetic. The first two calculations were used to compare the speed and precision of the algorithm with that of the direct calculation. The direct evaluation of the field in double precision was used as a standard for comparing the accuracies of the first two calculations. In all cases, we set $\epsilon = 10^{-8}$, and

$$m = n = 10 \cdot 2^k, \tag{66}$$

with k varying from 1 to 8.

Tables 1, 3, 5 contain the CPU timings for the examples 1, 2, 3 respectively. Following is a detailed description of the entries in these Tables.

n - the number of points at which the sum (1) is being evaluated.

T_{init} - the initialization time of the algorithm.

T_{alg} - the CPU time required by the algorithm once it has been initialized.

T_{dir} - the CPU time required by the direct calculation.

Tables 2, 4, 6 contain the accuracies for the examples 1, 2, 3 respectively. In the description of the entries of these tables below, S_k denotes the sum (1) at the point x_k as evaluated directly in double precision. S_k^{dir} denotes the sum (1) at the point x_k as evaluated directly in single precision, and S_k^{alg} denotes the sum (1) at the point x_k as evaluated in single precision via the algorithm of the present paper. Following is a detailed description of the entries in the Tables 2, 4, 6.

n - the number of points at which the sum (1) is being evaluated.

δ_{alg}^{max} - the maximum error produced by the algorithm at any point. It is defined by the formula

$$\delta_{alg}^{max} = \max_{1 \le k \le n} \mid S_k^{alg} - S_k \mid . \tag{67}$$

δ_{dir}^{max} - the maximum error produced by the direct calculation at any point. It is defined by the formula

$$\delta_{alg}^{max} = \max_{1 \le k \le n} \mid S_k^{dir} - S_k \mid . \tag{68}$$

$\delta_{alg}^{max,rel}$ - the maximum relative error produced by the algorithm at any point. It is defined by the formula

$$\delta_{alg}^{max,rel} = \max_{1 \le k \le n} \frac{\mid S_k^{alg} - S_k \mid}{\mid S_k \mid}. \tag{69}$$

$\delta_{dir}^{max,rel}$ - the maximum relative error produced by the direct calculation at any point. It is defined by the formula

$$\delta_{alg}^{max} = \max_{1 \le k \le n} \frac{\mid S_k^{dir} - S_k \mid}{\mid S_k \mid}. \tag{70}$$

δ_{alg}^{rel} - the relative error as defined in Section 3 as produced by the algorithm. It is given by the formula

$$\delta_{alg}^{rel} = \frac{\sum_{k=1}^{n} \mid S_k^{alg} - S_k \mid}{\sum_{k=1}^{n} \mid S_k \mid}. \tag{71}$$

δ_{dir}^{rel} - the relative error as defined in Section 3 as produced by the direct calculation. It is given by the formula

$$\delta_{dir}^{rel} = \frac{\sum_{k=1}^{n} \mid S_k^{dir} - S_k \mid}{\sum_{k=1}^{n} \mid S_k \mid}. \tag{72}$$

Following is a detailed description of the three examples.

Example 1. In this example, the points $\beta_1, \beta_2, \cdots, \beta_m$ and $x_1, x_2, \cdots, x_n$ were defined by the formulae

$$\beta_i = \frac{5}{m-1} \cdot (i-1), \tag{73}$$

$$x_k = \frac{5}{n-1} \cdot (k-1), \tag{74}$$

and the weights $\alpha_1, \alpha_2, \cdots, \alpha_m$ were generated randomly on the interval $[0, 1]$. Here, by "direct algorithm" we mean a straightforward implementation of the formula (17). The results of this set of experiments are summarized in Tables 1, 2.

Example 2. In this example, the points $\beta_1, \beta_2, \cdots, \beta_m$ and $x_1, x_2, \cdots, x_n$ were generated randomly on the interval $[0, 5]$, and the weights $\alpha_1, \alpha_2, \cdots, \alpha_m$ were generated randomly on the interval $[0, 1]$. Again, by "direct algorithm" we mean a straightforward implementation of the formula (17). The results of this set of experiments are summarized in Tables 3, 4.

Example 3. Here, we evaluate a polynomial of order n at a collection of randomly generated points on the interval $[0, 1]$. The coefficients of the $n-th$ order polynomial are randomly distributed on the interval $[0, 1]$. In this example, the direct evaluation of the polynomials is performed via the Horner's rule (see, for example, [3]), and the algorithm of this paper is applied via the formula (1). The results of this set of experiments are presented in Tables 5, 6.

The following observations can be made from the Tables 1-6, and are in agreemant with the results of our more extensive experiments.

1. In all cases, the accuracy produced by the algorithm of the present paper is comparable to that obtained by the direct calculation. For large n, the algorithm tends to be slightly more accurate.

2. The CPU times and accuracies produced by the algorithm are virtually independent of the distributions of points α_i, β_i, x_k in R^1.

3. When used for evaluating expressions of the form (17), the algorithm becomes faster than the direct calculation at $n = m \leq 20$, if the initialization time is ignored. If we include the initialization time, the break-even point is between $n = m = 40$ and $n = m = 60$.

4. When used for evaluating polynomials, the algorithm becomes faster than the direct calculation at roughly $n = m = 40$, if the initialization time is ignored. If we include the initialization time, the break-even point is roughly $n = m = 300$.

References

[1] A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison Wesley, Reading, Ma, 1974.

[2] A. Borodin, I. Munro, *The Computational Complexity of Algebraic and Numeric Problems,* Elsevier, New York, NY, 1975.

[3] G. Dahlquist, A. Bjork, *Numerical Methods,* Prentice-Hall, Englewood Cliffs, NJ, 1974.

Table 1

Example 1 : Timings

n	T_{init}	T_{alg}	T_{dir}
20	0.0112	0.0015	0.0081
40	0.0369	0.0042	0.0318
80	0.0802	0.0092	0.1278
160	0.136	0.0165	0.5202
320	0.218	0.0283	2.069
640	0.333	0.0468	8.368
1280	0.484	0.0784	33.25
2560	0.727	0.137	133.58

Table 2

Example 1: Accuracies

n	δ_{alg}^{max}	δ_{dir}^{max}	$\delta_{alg}^{max,rel}$	$\delta_{dir}^{max,rel}$	δ_{alg}^{rel}	δ_{dir}^{rel}
20	.412E-06	.638E-06	.383E-06	.243E-06	.359E-07	.556E-07
40	.179E-05	.219E-05	.459E-06	.418E-06	.783E-07	.960E-07
80	.408E-05	.688E-05	.623E-06	.671E-06	.100E-06	.169E-06
160	.138E-04	.262E-04	.825E-06	.116E-06	.173E-06	.326E-06
320	.378E-04	.873E-04	.597E-06	.195E-05	.238E-06	.550E-06
640	.922E-04	.231E-03	.103E-05	.258E-05	.279E-06	.699E-06
1280	.273E-03	.740E-03	.841E-06	.473E-05	.420E-06	.114E-05
2560	.522E-03	.233E-02	.880E-06	.886E-05	.407E-06	.181E-05

Table 3

Example 2 : Timings

n	T_{init}	T_{alg}	T_{dir}
20	0.0097	0.0011	0.0083
40	0.0275	0.0033	0.0332
80	0.0768	0.0089	0.1328
160	0.126	0.0157	0.536
320	0.210	0.0271	2.12
640	0.326	0.0455	8.50
1280	0.497	0.0784	34.12
2560	0.698	0.1351	138.34

Table 4

Example 2: Accuracies

n	δ_{alg}^{max}	δ_{dir}^{max}	$\delta_{alg}^{max,rel}$	$\delta_{dir}^{max,rel}$	δ_{alg}^{rel}	δ_{dir}^{rel}
20	.312E-06	.164E-06	.160E-06	.192E-06	.268E-07	.141E-07
40	.109E-05	.270E-05	.405E-06	.316E-06	.501E-07	.124E-06
80	.354E-05	.364E-05	.658E-06	.860E-06	.832E-07	.857E-07
160	.835E-05	.161E-04	.860E-06	.845E-06	.100E-06	.194E-06
320	.198E-04	.298E-04	.974E-06	.158E-06	.115E-06	.173E-06
640	.606E-04	.128E-03	.824E-06	.288E-05	.185E-06	.389E-06
1280	.336E-03	.657E-03	.914E-06	.423E-05	.207E-06	.100E-05
2560	.451E-03	.149E-02	.827E-06	.799E-05	.348E-06	.115E-05

Table 5

Example 3 : Timings

n	T_{init}	T_{alg}	T_{dir}
20	0.0135	0.0015	0.0013
40	0.0435	0.0052	0.0047
80	0.0948	0.0109	0.0179
160	0.1327	0.0172	0.0729
320	0.222	0.0286	0.2779
640	0.306	0.0445	1.101
1280	0.422	0.0718	4.54
2560	0.664	0.1322	18.37

Table 6

Example 3: Accuracies

n	δ_{alg}^{max}	δ_{dir}^{max}	$\delta_{alg}^{max,rel}$	$\delta_{dir}^{max,rel}$	δ_{alg}^{rel}	δ_{dir}^{rel}
20	.772E-06	.260E-05	.301E-06	.305E-06	.673E-07	.227E-06
40	.218E-05	.275E-05	.396E-06	.337E-06	.955E-07	.121E-06
80	.518E-05	.554E-05	.528E-06	.210E-06	.127E-06	.136E-06
160	.615E-05	.462E-05	.671E-06	.304E-06	.766E-07	.576E-07
320	.103E-04	.232E-05	.851E-06	.387E-06	.646E-07	.146E-06
640	.224E-04	.295E-04	.832E-06	.422E-06	.680E-07	.894E-07
1280	.615E-04	.360E-03	.745E-06	.120E-05	.949E-07	.555E-06
2560	.757E-04	.120E-02	.862E-06	.191E-05	.590E-07	.932E-06