JOURNAL OF APPROXIMATION THEORY 4, 332–338 (1971)

Running Orthogonalization*

JOHN R. RICE

Division of Mathematical Sciences, Purdue University, Lafayette, Indiana 47907

Received February 16, 1970

1. INTRODUCTION

This paper is motivated by the following

PROBLEM. *Given data* (f_j, t_j), $j = 1, 2, \ldots, K$, *basis functions* $\varphi_i(t)$, $i = 1, 2, \ldots, n$, *and* $\epsilon > 0$, *determine the index* M *so that the least squares approximation to* (f_j, t_j), $j = 1, 2, \ldots, M$, *(by the functions* $\varphi_i(t)$*) has an error* $\leqslant \epsilon$ *and* M *is the largest index for which this is true.*

The obvious solution is to successively compute the least squares approximations to (f_j, t_j), $j = 1, 2, \ldots, L$, and increase L until the error requirement is violated. This solution leads, however, to a lengthy computation and, hence, another process, running orthogonalization, is presented here which is an order of magnitude more efficient.

Running orthogonalization is developed in a somewhat more abstract form in the next section. The idea is similar in philosophy to updating the inverse matrix (as in the simplex method), but the mechanics are not similar. The efficiency of running orthogonalization is analyzed in the third section and the results of a comparison of actual computations using several schemes are presented. It is seen that running orthogonalization requires about twice the computation as would be required if the value of M were known a priori and one computed the least squares approximation on the first M data points by an ordinary orthogonalization scheme.

2. RUNNING ORTHOGONALIZATION

We consider functions defined on a domain

$$T = T_1 \cup T_2$$

* This work was partially supported by NSF grants GP-07163 and GP-05850.

and an inner product denoted by (x, y). The restrictions of functions to T_j are denoted by superscript j, i.e.,

$$f^1(t) = \begin{cases} f(t), & t \in T_1, \\ 0, & t \in T_2, \end{cases}$$

$$f^2(t) = \begin{cases} 0, & t \in T_1, \\ f(t), & t \in T_2. \end{cases}$$

Inner products $(f, g)_j$ are defined on T_j, $j = 1, 2$, and are related by:

$$(f, g) = (f^1, g^1)_1 + (f^2, g^2)_2. \tag{2.1}$$

The problem considered may be phrased as follows: Given a set $\{\varphi_i^1 \mid i = 1, 2, \ldots, n\}$, orthonormal on T_1, and extensions φ_i^2, determine an orthonormal set $\{\psi_i \mid i = 1, 2, \ldots, n\}$, equivalent to $\{\varphi_i\}$ (i.e., the linear spans in $C(T)$ of $\{\psi_i\}$ and $\{\varphi_i\}$ are the same.) Further, given a function $f(t)$, its coefficients a_i^1 of best least squares approximation by the φ_i^1 and the associated error d^1, determine the corresponding quantities for approximation by the ψ_i. The idea is to incorporate the fact that the φ_i^1 are already orthonormal on T_1 into the determination of the ψ_i. The running orthogonalization procedure presented here to accomplish this is a version of the Gram-Schmidt process.

The final formulas are presented below. The reader may verify that the ψ_{i_2} given are in fact orthonormal and that the formules for the coefficients a_i^2 and the associated error d^2 are correct. We omit the superscripts 1 and 2 in the inner products where no ambiguity is possible. The process is initialized by

$$\psi_i = \varphi_1 / \| \varphi_1 \|, \qquad \| \varphi_1 \|^2 = 1 + (\varphi_1, \varphi_1)_2, \tag{2.2}$$

and is continued, for $k = 2, 3, \ldots, n$, as follows:

$$\psi_k^* = \varphi_k - \sum_{j=1}^{k-1} (\varphi_k, \psi_j)_2 \psi_j, \tag{2.3}$$

$$\psi_k = \psi_k^* / \| \psi_k^* \|, \qquad \| \psi_k^* \|^2 = 1 + (\varphi_k, \varphi_k)_2 - \sum_{j=1}^{k-1} (\varphi_k, \psi_j)_2^2. \tag{2.4}$$

The coefficients and error are determined by

$$a_k^2 = \left[a_k^1 + (f, \varphi_k)_2 - \sum_{j=1}^{k-1} (\varphi_k, \psi_j)_2 a_j^2 \right] \Big/ \| \psi_k^* \|, \tag{2.5}$$

$$d^2 = \left[d^1 + (f, f)_2 - \sum_{j=1}^{n} (a_j^2)^2 \right]^{1/2} \tag{2.6}$$

The basis for the efficiency of running orthogonalization is the fact that these formulas contain only inner products on T_2. Furthermore, the number of inner products is the usual number for orthonormalization schemes.

The normal use of such a scheme involves a fixed basis throughout. Functions may be represented as vectors of values in case T is a finite set. The effect on efficiency of the choice of representation is discussed in the next section. The preceding formulas may be restated in terms of a fixed set of basis functions (e.g., powers of x, orthogonal polynomials, trigonometric functions); we present these formulas which are somewhat tedious to obtain.

Let $\{P_i \mid i = 1, 2,..., n\}$ be the fixed basis and let

$$\begin{aligned} \varphi_i(t) &= \sum_{j=1}^{n} r_{ij} P_j(t), \qquad i = 1, 2,..., n, \\ \psi_i(t) &= \sum_{j=1}^{n} s_{ij} P_j(t), \qquad i = 1, 2,..., n, \end{aligned} \tag{2.7}$$

be representations of the orthonormal sets $\{\varphi_i\}$ and $\{\psi_i\}$. In practice, one usually has $r_{ij} = s_{ij} = 0$ for $j > i$. The process is initialized by

$$\begin{aligned} &p_{11} = (\varphi_1, \varphi_1)_2, \qquad q = \sqrt{1 + p_{11}}, \\ &a_1^2 = [a_1^1 + (f, \varphi_1)_2]/q, \\ &s_{1j} = r_{1j}/q, \qquad j = 1, 2,..., n. \end{aligned} \tag{2.8}$$

It is continued for $k = 2, 3,..., n$ as follows:

$$\begin{aligned} &p_{kj} = (\varphi_k, \psi_j)_2, \quad j = 1, 2,..., k-1, \qquad p_{kk} = (\varphi_k, \varphi_k)_2, \\ &q = \left[1 + p_{kk} - \sum_{j=1}^{k-1} p_{kj}^2\right]^{1/2} \\ &a_k^2 = \left[a_k^1 + (f, \varphi_k)_2 - \sum_{j=1}^{k-1} p_{kj} a_j^2\right]\Big/q, \\ &s_{kj} = \left(r_{kj} - \sum_{l=1}^{k-1} p_{kl} s_{lj}\right)\Big/q, \qquad j = 1, 2,..., n. \end{aligned} \tag{2.9}$$

Finally, we have

$$d^2 = \left[d^1 + (f, f)_2 - \sum_{j=1}^{n} (a_j^2)^2\right]^{1/2}. \tag{2.10}$$

We note that almost identical formulas are valid in case

$$T = T_1 - T_2$$

and

$$(f, g) = (f^1, g^1)_1 - (f^2, g^2)_2 .$$

The formulas (2.2) through (2.6) need only be modified by changing the sign of the coefficients of the inner products on T_2. The sum of squares terms in (2.4) do not, of course, change sign. Further simple modifications cover the case where $T = (T_1 \cup T_2) - T_3$.

3. Implementation Aspects

The primary objective of this algorithm is to be efficient and thus it is appropriate to compare it with alternative schemes. The classical approach is to "count" the number of arithmetic operations required by an algorithm. The shortcomings of this approach are well known, but it nevertheless may be a useful guide. Later in this section we also present actual timing comparisons and some remarks on accuracy and stability.

We *assume* that all the orthogonalization schemes considered are carried out in the context of a fixed set of basis functions as in (2.7) and that T is a finite set. A key factor in the computation is then the work of evaluation of the inner products and, in turn, the evaluation of the basis functions $P_j(t)$. We consider three distinct cases:

A. $\varphi_j(t)$ and $\psi_j(t)$ are polynomials in t of exact degree j and are evaluated by j additions and multiplications whenever a value is required.

B. $\varphi_j(t)$ and $\psi_j(t)$ are as in case A except that, when (φ_k, ψ_j) is required for several values of j (typically, $j = 1, 2, \ldots, k$), then the values of $\varphi_k(t)$ are computed and temporarily stored during this computation.

C. $\varphi_j(t)$ and $\psi_j(t)$ are given by tables of values and thus may be "evaluated" with no computation.

These three cases do not cover all possibilities, but are common ones. In particular, they include the well-known trade-off between computational speed and storage space in actual computation. This effect is illustrated by the following simple

PROBLEM. *Compute the best least-squares approximation with N basis functions where T has L points.*

If one assumes appropriate equivalences between different arithmetic operations, one has the comparison seen in Table 1.

TABLE 1

Illustration of the Trade-off Between Computational Work and Storage Used in a Simple Least Squares Problem

	Units of computation	Storage required
Case A	$\frac{LN^3}{2} + 2LN^2 + \frac{N^3}{6} - 2N^2$	N^2
Case B	$\frac{LN^3}{6} + 2LN^2 + \frac{N^3}{6} - 2N^2$	$N^2 + L$
Case C	$LN^2 + 4LN + \frac{N^3}{6} - 2N^2$	NL

Lower order terms have been neglected and the storage required refers only to data for the representation of the functions involved. (Careless organization will lead to $2NL$ for Case C).

We now compare the computational efforts required by three different methods to compute the best least squares approximation to a function on $n + M + 1$ points with polynomials of degree n. The methods are:

Method 1. Compute the best approximation on the set of L points t_j, $j = 1, 2, \ldots, L$, for $L = n + 1$ to $n + M + 1$.

Method 2. Use running orthogonalization starting with $T_1 = \{t_1, \ldots, t_{m+1}\}$ and successively taking $T_2 = \{t_j\}$ with $j = n + 2, \ldots, n + M + 1$.

Method 3. Compute the best approximation on the $n + M + 1$ points only using Gram–Schmidt orthogonalization.

Recall that in the application envisaged one cannot use Method 3 as the value of M is not known. The units of computation required are tabulated below for the nine combinations of methods and cases of representation. The units are estimated on the basis of an examination of actual algorithms (in Fortran), but they are only approximate due to the various assumptions made. The most noteworthy fact seen in Table 2 is that the expressions which Method 2 yields are linear in M while those obtained by Method 1 are quadratic. Indeed, the "overhead" for running orthogonalization is rather modest compared to direct computation except for Case C and large values of n.

TABLE 2

Estimates of Units of Computation for Various Combinations of Methods of Computation and Cases of Representation

	Method 1	Method 2	Method 3
Case A	$\frac{n^3M^2}{4} + \frac{n^4M}{2} + \frac{7n^2M^2}{4}$	$\frac{5n^3M}{6} + 4n^2M + \frac{n^4}{2}$	$\frac{n^3M}{2} + \frac{7n^2M}{2} + \frac{n^4}{2}$
Case B	$\frac{n^3M^2}{12} + \frac{7n^2M^2}{4} + \frac{n^4M}{6}$	$\frac{n^3M}{2} + 3n^2M + \frac{n^4}{6}$	$\frac{n^3M}{6} + 2n^2M + \frac{n^4}{6}$
Case C	$\frac{n^2M^2}{2} + 3nM^2 + n^3M$	$\frac{n^3M}{3} + 3n^2M + n^3$	$n^2M + 4nM + n^3$

To further compare the efficiency of these methods, a large number of actual computations have been timed. All programs used Case A in the evaluation of the basis functions. The results of these computations are summarized below.

(a) Running orthogonalization requires from 1.5 (for $M = 5$) to 2.1 (for $M > 100$) as much time as ordinary Gram–Schmidt orthogonalization (Method 3). These factors are fairly independent of n for $n \leqslant 8$. The two programs compared are as identical as possible except for the orthogonalization scheme used.

(b) Two programs which implement Gram–Schmidt orthogonalization in apparently identical manners differ by a factor of 2 in computation time required. The reason for this factor can be found upon a detailed examination, but "mathematically" the programs are the "same".

(c) Two programs were compared which use Gram–Schmidt and the Forsythe scheme based on the three term recurrence relation. All other factors are as identical as possible. The Forsythe scheme is relatively more efficient for high degrees. The factors of improvement observed are 1.1, 1.3, 1.4, 1.6 for $m = 2, 4, 6, 8$, respectively.

We may summarize this analysis and these experiments by saying that running orthogonalization gives a very significant improvement in computation efficiency compared to Method 1 and is fairly competitive with Method 3. The difference between Methods 2 and 3 is no more than that caused by seemingly trivial differences in different implementations of Method 3.

Several tests were made to see if running orthogonalization is numerically stable. With M up to 400 and n up to 8, no deterioration was observed in the accuracy. If anything, running orthogonalization gives more accurate results, especially for high polynomial degrees.