Algorithms

L.D. Fosdick
Editor

Submittal of an algorithm for consideration for publication in Communications of the ACM implies that unrestricted use of the algorithm within a computer is permissible.

Algorithm 414

Chebyshev Approximation of Continuous Functions by a Chebyshev System of Functions [E2]

G.H. Golub and L.B. Smith* (Recd. Oct. 11, 1967, Jan. 27, 1969, and Apr. 11, 1970) Dept. of Computer Science, Stanford University, Stanford CA 94305

Key Words and Phrases: approximation, Chebyshev approximation, Remez algorithm
CR Categories: 5.13

The second algorithm of Remez can be used to compute the minimax approximation to a function, $f(x)$, by a linear combination of functions, $\{Q_i(x)\}_0^n$, which form a Chebyshev system. The only restriction on the function to be approximated is that it be continuous on a finite interval $[a,b]$. An Algol 60 procedure is given, which will accomplish the approximation. This implementation of the second algorithm of Remez is quite general in that the continuity of $f(x)$ is all that is required whereas previous implementations have required differentiability, that the end points of the interval be "critical points," and that the number of "critical points" be exactly $n + 2$. Discussion of the method used and of its numerical properties is given as well as some computational examples of the use of the algorithm. The use of orthogonal polynomials (which change at each iteration) as the Chebyshev system is also discussed.

Description

1. *Introduction.* Given a Chebyshev system, $\varphi_0(x), \varphi_1(x), \ldots, \varphi_n(x)$, we define the Chebyshev or minimax approximation to a continuous function $f(x)$ over an interval $[a, b]$ to be the function

$$P_n(x) = c_0\varphi_0(x) + \cdots + c_n\varphi_n(x), \tag{1.1}$$

such that ϵ is minimized, where

$$\epsilon = \max_{a\le x\le b} |f(x) - P_n(x)|. \tag{1.2}$$

If $\varphi_i(x) = x^i$, we have the minimax polynomial approximation of degree n to $f(x)$. If $\varphi_i(x) = T_i(x)$, where $T_i(x)$ denotes the Chebyshev polynomial of the first kind of order i, we have the minimax approximation as a sum of Chebyshev polynomials. For the definition of a Chebyshev system, see Achieser [3, p. 73].

* Now at Computing Center, University of Colorado, Boulder, CO 80302

The algorithm presented here computes the coefficients c_i, $i = 0, 1, \ldots, n$, in (1.1) for any given Chebyshev system $\varphi_i(x)$, $i = 0, 1, \ldots, n$. The algorithm is based on the second algorithm of Remez [1], and also makes use of the exchange method described by Stiefel [2].

The characterization of the error curve, given by

$$\epsilon(x) = \sum_{i=0}^{n} c_i\varphi_i(x) - f(x), \tag{1.3}$$

is the basis for the second algorithm of Remez. It is shown, for example, by Rice [11, p. 56] that $p_n^*(x) = \sum_{i=0}^{n} c_i\varphi_i(x)$ is the Chebyshev approximation to $f(x)$ on $[a, b]$ if and only if there exists a set of points $a \le x_0 < x_1 < x_2 < \cdots < x_{n+1} \le b$ such that

(a) $\epsilon(x_{i+1}) = -\epsilon(x_i)$,
(b) $|\epsilon(x_i)| = \epsilon^*$, and
(c) $\max_{a\le x\le b} |\epsilon(x)| = \epsilon^*$.

Thus, when the computed error curve attains this "equal ripple' character with at least $n + 1$ sign changes in $[a,b]$ we know we have the desired minimax approximation.

The second algorithm of Remez, based on the characterization, can be outlined in three steps.

(i) Choose an initial set of points, the reference set, $a \le x_0 < x_1 < \cdots < x_{n+1} \le b$.
(ii) Compute the discrete Chebyshev approximation to $f(x)$ on the reference set.
(iii) Adjust the points of the reference set to be the extrema of the error curve (1.3).

Steps (ii) and (iii) are repeated until convergence is obtained.

Proof of the existence of the minimax polynomial (given by (1.1) and (1.2) with $\{\varphi_i\}_0^n$, a Chebyshev system) is given by Achieser [3, p. 74].

Proof that the second algorithm of Remez converges for any starting values for the critical points is given by Novodvorskii and Pinsker [4]. If $f(x)$ is differentiable, Veidinger [12] proves that the convergence is quadratic. That is

$$\epsilon^* - \epsilon^{(k)} = O(\epsilon^* - \epsilon^{(k-1)})^2, \text{ as } k \to \infty,$$

where ϵ^* is the maximum error for the Chebyshev approximation and $\epsilon^{(k)}$ is the maximum error at the kth iteration. A survey article concerned with minimax approximations is given by Fraser [8].

2. *Applicability.* The algorithm presented herein has wide applicability in that it can be used to approximate any continuous function given on an arbitrary closed interval. In addition, the

approximating function is not restricted to polynomials or Chebyshev polynomials, but is allowed to be any linear Chebyshev system to be supplied by the user. Three simplifying assumptions often made in an implementation of the second algorithm of Remez are:

(a) Differentiability of $f(x)$, the function to be approximated. (see [6], for example)
(b) The end points of the interval are critical points (see [8, p. 299]).
(c) The existence of exactly $n + 2$ points of extreme value on the error curve (see [8, p. 299]).

None of these three assumptions is made for this algorithm.

3a. Formal parameter list: input to the procedure

n integer degree of the Chebyshev system of functions to be used in the fit $\{\varphi_0(x), \varphi_1(x), \cdots, \varphi_n(x)\}$.

a lower end point of the interval of approximation, of type real.

b upper end point of the interval of approximation, of type real.

kstart integer controlling the number of points ($kstart \times (n+2)$) used in the initial approximation. See (i) in Section 5.

kmax integer allowing control of the number of times k is increased above *kstart*.

loops integer allowing control over the number of iterations taken by Remez's second algorithm if convergence is not yet attained.

f a real procedure to compute the function $f(x)$ to be approximated; procedure heading required:

```
real procedure f(x);
value x;
real x;
```

the argument is the untransformed variable x. $f(x)$ must be continuous in the interval $[a, b]$.

chebyshev a procedure to evaluate the Chebyshev system of functions being used at some point, x, in the interval $[a, b]$; procedure heading required:

```
procedure chebyshev(n, x, t);
value n, x;
integer n;
real x;
real array t;
```

n is the degree of the system, x is the point in $[a, b]$, and t is an array that will contain the values $t[i] = \varphi_i(x)$, $i = 0, 1, \ldots, n$.

eps a real procedure to compute the error curve given by (5.1); procedure heading required:

```
real procedure eps(x, c, n);
value x, n;
real x;
integer n;
real array c;
```

x is a point in $[a, b]$, n is the degree of the system, and c is an array containing the coefficients of the approximation, $c[i] = c_i$ in (5.1).

exchange a procedure, [10] for example, to locate the $n + 2$ subset of $m + 1$ given points which determine the minimax polynomial on those $m + 1$ points; procedure heading required:

```
procedure exchange (a,d,c,m,n,refset,emax,singular,r);
value m,n; integer m,n; real emax;
real array a,d,c,r;
integer array refset;
label singular;
```

a is a real $m + 1$ by $n + 2$ array, d is a $m + 1$ component vector, c is a $n + 2$ component vector, $m + 1$ is the integer number of points $(x_0, \ldots, x_m)$, n is the degree of the system, *refset* is a $n + 2$ component integer vector, *emax* is a real number and *singular* is a label. r is a vector containing the $m + 1$ values of the residual at the $m + 1$ points under consideration. On entry the components of a and d are

$a[i,j] = \varphi_j(x_i)$ and

$d[i] = f(x_i), \quad i = 0(1)m, \quad j = 0(1)n.$

Upon exit from *exchange*, the array c contains the coefficients of the minimax function found, *refset* contains the subscripts identifying the points used to compute the minimax function, i.e. the reference set, and *emax* contains the value of the maximum deviation of the minimax function from $f(x)$ on the points x_i, $i = 0(1)m$.

3b. Formal parameter list: output from the procedure

c the array of coefficients c_i of eq. (5.1).

emax the maximum modulus of the error curve (5.1) for the final approximation function, of type real.

trouble a label to which control is transferred if *remez* does not converge properly.

why an integer whose value on exit will be set to one of the following:

why = −1 if number of added points is greater than n. (See step (ii) in Section 5.)
why = 1 if trouble occurs in procedure *quadraticmax*.
why = 2 if trouble occurs in procedure *exchange*.
why = 3 if no convergence after iterating *loops* times.
why = 4 converged according to the maximum and minimum residual comparison.
why = 5 converged according to *why* = 4 and the critical point test.
why = 6 converged according to *why* = 4 and the coefficient test.
why = 7 converged according to *why* = 4 and both the critical point and the coefficient tests.
why = 8 converged according to critical point test only.
why = 9 converged according to coefficient test only.
why = 10 converged according to critical point and coefficient tests.

4. *Organization and notational details.* The algorithm calls for three procedures, in addition to the function $f(x)$ to be approximated, as indicated by the formal parameter list.

exchange Based on Stiefel's Exchange algorithm, which finds the $n + 2$ subset of $m + 1$ given points which determine the minimax polynomial. Use [10], for example.

eps To be supplied by user: *eps* computes the error curve

$$\epsilon(x) = \sum_{i=0}^{n} c_i\varphi_i(x) - f[x] \tag{4.1}$$

where the c_i, $i = 0, \ldots, n$, are parameters and the $\varphi_i(x)$, $i = 0, 1, \ldots, n$, are the Chebyshev system of functions being used to fit the function $f(x)$. For best results $\epsilon(x)$ should be computed in double precision and then rounded to single precision accuracy. If $f(x)$ cannot be calculated easily or efficiently in double precision at least the sum, $\sum_{i=0}^{n} c_i\varphi_i(x)$, should be accumulated in double precision and rounded to single.

chebyshev To be supplied by user: *chebyshev* evaluates the Chebyshev system $\varphi_i(x)$, $i = 0, 1, \ldots, n$ for a given argument x. *chebyshev* is called by *eps*.

The functions $\epsilon(x)$ and $\varphi_i(x)$ (computed by *eps* and *chebyshev*) can often be computed by simple recursive procedures. For example, if the Chebyshev system used is the set of Chebyshev polynomials, there is a well-known recurrence relation $(\varphi_{i+1}(x) = 2x\varphi_i(x) - \varphi_{i-1})$ that can be used to efficiently evaluate the required functions.

An outline of the organization of the algorithm is given in the following steps:

(i) Let $m = k \times (n+2)$, take $m + 1$ points in the interval $[a,b]$ and use *exchange* to determine the "best" polynomial (i.e. the

$$c_i \ni \max_{0 \le j \le n} \left| \sum_{i=0}^{n} c_i\varphi_i(x_j) - f(x_j) \right| = \text{minimum})$$

on those points. Exchange will pick $n + 2$ of the original points as

critical points. The $m + 1$ points are chosen equally spaced or as the zeros of $T_{m-1}(x) - T_{m-3}(x)$ with $k \geq 1$.
(ii) Use the $n + 2$ points chosen by *exchange* in step (i) and ν other local extrema (subject to the conditions discussed under Example 2, Section 6) as input to the procedure *quadraticmax* ($\nu \geq 0$).
(iii) Procedure *quadraticmax* adjusts the $n + \nu + 2$ critical points to be the abscissas of the extrema of the error curve given by (4.1). Section 5b gives a discussion of how the adjustments are computed. After adjustment the new points are tested for alternation of sign, and if the property has been lost, we increase k and go back to step (i).
(iv) The adjusted critical points are then input to *exchange* which finds the new coefficients c_i, $i = 0, 1, \cdots, n$ for the "best" polynomial on the adjusted $n + \nu + 2$ points.
(v) Now convergence tests can be applied to the coefficients c_i, found in step (iv), to the critical points x_i, $i = 0, 1, \cdots, n$ and to the extreme values of (4.1). If not converged, go back to step (iii) since the previous critical points will not be the exact extreme points after the approximating polynomial is changed in step (iv).

5a. *Discussion of numerical properties and methods: accuracy and convergence.* The accuracy of the approximations generated by this procedure is limited by the precision of the arithmetic used and the accuracy of the subsidiary procedures *f*, *exchange*, *eps*, and *chebyshev*. The use of double precision in *eps*, for example, can improve the results of *remez* since it will then have a "smoother" error curve to work on. This use of double precision in *eps* is strongly recommended by the authors. The maximum absolute error of the approximation is output from *remez* and depends, of course, on n, the degree of approximation.

The procedure is deemed to have converged when the coefficients of the approximating function or the critical points have satisfied certain relative criteria between successive iterations. We use the notation $c_i^{(n)}$ to represent the ith coefficient at the nth iteration and similarly, $x_i^{(n)}$ represents the ith critical point at the nth iteration.

When

$$\max_i |c_i^{(n)} - c_i^{(n-1)}| \leq epsc\,|c_i^{(n)}| \tag{5.1}$$

or

$$\max_i |x_i^{(n)} - x_i^{(n-1)}| \leq epsx\,|x_i^{(n)}| \tag{5.2}$$

we consider the procedure to have converged. If $|c_i^{(n)}|$ or $|x_i^{(n)}|$ is very small the relative test is not appropriate. In that case we test $|c_i^{(n)} - c_i^{(n-1)}|$ and $|x_i^{(n)} - x_i^{(n-1)}|$ against allowed absolute errors, *absepsc* and *absepsx*. Typical values for the constants (for an 11-decimal place machine) could be

$$\begin{aligned} epsc &= 10^{-8} \\ epsx &= 10^{-4} \\ absepsc &= 10^{-8} \\ absepsx &= 10^{-4} \end{aligned} \tag{5.3}$$

A third convergence criterion is the comparison of the maximum and minimum magnitudes of the error curve at the critical points. Let

$$maxr = \max_i |\epsilon(x_i^{(n)})|$$

and

$$minr = \min_i |\epsilon(x_i^{(n)})|$$

where $\{x_i^{(n)}\}$ are the critical points chosen at the nth iteration, and then make the following test. If $maxr \leq rcompare \times minr$ then claim convergence. A typical value for the constant *rcompare* could be 1.0000005.

When the maximum absolute error approaches $10^{-s}(f_m)$, where s is the number of places available in the machine, and f_m is $max_{a \leq x \leq b} |f(x)|$, we are approaching the limit of obtainable accuracy. We are working with

$$\epsilon(x) = P_n(x) - f(x) \tag{5.4}$$

so when $\epsilon(x)$ is nearly equal to $10^{-s}f(x)$, we are losing about s places in the subtraction in (5.4). This is where judicious use of double precision can be made to increase accuracy if necessary. $P_n(x)$ can be computed in double precision and a single precision difference formed, or for even further accuracy $f(x)$, if possible, could be computed in double precision and the double precision difference taken.

A comparison of the discrete approximation on a finite number of points in an interval, and the continuous approximation which this algorithm finds, is studied by Rivlin and Cheney in [9]. Rice [11, pp. 66–70] discusses the question of convergence (and rate of convergence) of the discrete approximation to the continuous approximation. This relates to the question of how large to choose k in step (i), Section 4. We have found that for well-behaved functions like e^x on $[-1,1]$ a value for k of about 3 gives good starting values. On the other hand a function like $1/(x-\lambda)$ on $[-1,1]$ with $\lambda > 1$ and λ near 1 requires k to be about 15 to obtain good starting values. The choice of k should be large enough so that the initial approximation chosen by the procedure *exchange* is close enough to the final approximation to insure that the "alternation of sign" property is never lost during the iterations. There is no known method of choosing such a k a priori. This is why the algorithm tests for "alternation of signs" at each iteration and increases k if the property is lost.

5b. *Discussion of numerical properties and methods: Locating the extrema of* $\epsilon(x)$. Most of the programming effort is involved in locating the extrema of the error function $\epsilon(x)$. The programming is similar to that done by C.L. Lawson in a *Fortran* program to compute the best minimax approximation [7]. $\epsilon(x)$ is given by

$$\epsilon(x) = \sum_{i=0}^{n} c_i\varphi_i(x) - f(x).$$

The procedure *exchange* then is used to compute the coefficients of the minimax function. That is, given $n + \nu + 2$ points, $\nu \geq 0$, *exchange* computes the coefficients of the function $\sum_{i=0}^{n} c_i\varphi_i(x)$ such that on the discrete set of points $\epsilon(x_j)$, $j = 0, 1, \cdots, n + \nu + 1$ has at least $n + 2$ extreme values (at the given points) equal in magnitude and of alternating signs. The satisfaction of this condition when the points are indeed the extrema of the continuous $\epsilon(x)$ guarantees that $\sum_{i=0}^{n} c_i\varphi_i(x)$ is the unique minimax approximating function that we seek.

5b.1 *Discussion of numerical properties and methods: Parabolic approximation to locate extremum.* Given the initial guesses x_i, $i = 0, 1, \cdots, n + \nu + 1$ (at each iteration) for the abcissas of the extrema of the error curve, we must locate these critical points more precisely. We consider two cases. First the interior points, and secondly the least and greatest of the initial guesses which may be equal to the respective end points of the interval on which the function is to be approximated.

For interior points we do the following. Take

$$\begin{aligned} u &= x_i \\ v &= x_i + \alpha(x_{i+1} - x_i) \\ w &= x_i + \alpha(x_{i-1} - x_i) \end{aligned} \tag{5.5}$$

where α is a parameter $0 < \alpha < 1$ (e.g. $\alpha = 0.1$). We then determine the parabola through the three points $\epsilon(u)$, $\epsilon(v)$, and $\epsilon(w)$. The abscissa, x^*, corresponding to the vertex of this parabola is then taken as the next guess for the ith "critical point." The point x^* is given by

$$x^* = \frac{1}{2}\,\frac{[(u^2 - v^2)\epsilon(w) + (v^2 - w^2)\epsilon(u) + (w^2 - u^2)\epsilon(v)]}{[(u - v)\epsilon(w) + (v - w)\epsilon(u) + (w - u)\epsilon(v)]}. \tag{5.6}$$

For computational purposes x^* is not computed directly by (5.6)

since for u, v, and w very close, the denominator will be quite small. Therefore, the denominator of (5.6) is computed

$$d = [(u-v)\epsilon(w)+(v-w)\epsilon(u)+(w-u)\epsilon(v)], \tag{5.7}$$

and then by dividing out (5.6), we express x^* as

$$x^* = \begin{cases} \frac{1}{2}(u+v) & \text{if } d = 0 \\ \frac{1}{2}(u+v) + \frac{1}{2}\dfrac{(v-u)(u-w)[\epsilon(v)-\epsilon(w)]}{d} & \text{if } d \neq 0. \end{cases} \tag{5.8}$$

Once x^* is computed, it is then tested to insure acceptability since for u, v, and w very close, machine roundoff may introduce spurious results. Also, the value of α or the nature of the function $f(x)$ and therefore of $\epsilon(x)$ may introduce an unacceptable value for x^* in which case u, v, or w, whichever has highest ordinate value, is used for x^*. If x^* is acceptable it can replace u, v, or w, whichever has the lowest (in abolute value) ordinate value on the error curve $\epsilon(x)$, and a second x^* is computed. This iteration will converge to the abcissa of the extremum near x_i if roundoff is ignored and u, v, and w are sufficiently close to that point. (Compare convergence to Muller's method for solving algebraic equations [5].) However, this iteration need not be carried out excessively (2–4 iterations should be sufficient) since during each iteration of the overall process we recompute the approximating function and thereby obtain a new error curve whose extrema will not necessarily have the same abscissas.

For the end points (5.5) cannot apply since x_{i+1} and x_{i-1} do not exist at the right and left ends respectively. Therefore we take, at the left end for example,

$$\begin{aligned} u &= x_i \\ v &= x_i + \alpha(x_{i+1}-x_i) \\ w &= \begin{cases} x_i + \beta(x_{i+1}-x_i) & \text{if } x_i = a \\ x_i + \alpha(a-x_i) & \text{if } a < x_i, \end{cases} \end{aligned} \tag{5.9}$$

with the requirement that $\alpha \neq \beta$. The right end is handled similarly. Again the parabola through the three points $\epsilon(u)$, $\epsilon(v)$, and $\epsilon(w)$ is used to determine x^*. The tests for acceptability and iterations are performed as they were for the interior points.

5b.2 *Discussion of numerical properties and methods: Crude search to locate extremum.* In case approximation by parabola does not yield an acceptable value for the abscissa of an extremum, the following rather crude method works effectively. We simply divide the interval under consideration into l equal intervals (e.g. $l=10$) and examine the ordinate of the error curve at the end points of the intervals. The points to the left and right of the point with maximum ordinate (in absolute value) then define a new interval upon which the process is repeated. This subdivision continues until the subintervals become smaller than some specified value (e.g. 10^{-5}). The method causes the function to be evaluated more often than the parabolic approximation, but works successfully at a point where the error curve has a sharp cusp-like extremum.

The choice of $l = 10$ in this crude search procedure is arbitrary. In fact, for an initial interval of length I, a smaller value, say $l = 4$, would reduce the subinterval size to $10^{-5} \cdot I$ with a minimum of 21 function evaluations, whereas using $l = 10$ would require at least 51 function evaluations. However, small values of l increase the chances of missing the true extremum.

To decide whether to use this crude search or not we employ a relative test. Let the parabolic choice be x^* and the three points used to compute x^* be u, v, and w. Then one would expect (hope) that $|\epsilon(x^*)| \geq |\epsilon(u)|$, $|\epsilon(v)|$, and $|\epsilon(w)|$, in which case x^* has the desired properties. However, if $\epsilon_m = max_{x=u,v,w} |\epsilon(x)|$, and $|\epsilon(x^*)| < \epsilon_m$, then we must doubt the acceptability of x^* and perhaps use the crude method to determine x^*. We found a successful way to make this decision was to use the crude method if $||\epsilon(x^*)| - \epsilon_m| > C \cdot \epsilon_m$, where C is an arbitrary constant (e.g. 10^{-4}).

Fig. 1

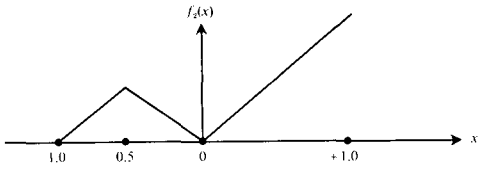

Table I. Coefficients c_i of "best" polynomial $P_4(x) = \sum_{i=0}^{4} c_i T_i(x)$ (to 6D)

i	Start	Iteration 1	Iteration 2	Iteration 3
0	1.266 063	1.266 066	1.266 066	1.266 066
1	1.130 321	1.130 318	1.130 318	1.130 318
2	0.271 495	0.271 495	0.271 495	0.271 495
3	0.044 337	0.044 336	0.044 336	0.044 336
4	0.005 523	0.005 519	0.005 519	0.005 519

Table II. Critical points, x_j, of best polynomial (to 6D)

j	Start	Iteration 1	Iteration 2	Iteration 3
0	−1.000 000	−1.000 000	−1.000 000	−1.000 000
1	−0.771 429	−0.797 573	−0.797 682	−0.797 682
2	−0.257 143	−0.278 189	−0.279 152	−0.279 152
3	0.314 286	0.339 805	0.339 061	0.339 061
4	0.828 571	0.820 978	0.820 536	0.820 536
5	1.000 000	1.000 000	1.000 000	1.000 000

Table III. Comparison of starting values x_j for $f(x) = e^x$, $n = 4$ (to 3D)

j	$T_5(x) - T_3(x) = 0$ or $\lvert T_5(x)\rvert = 1$	*exchange* on $6(N+2)$ points equally spaced	*exchange* on 201 points equally spaced	TRUE (computed)
0	−1.000	−1.000	−1.000	−1.000
1	−0.809	−0.771	−0.800	−0.798
2	−0.309	−0.257	−0.280	−0.279
3	0.309	0.314	0.340	0.339
4	0.809	0.829	0.820	0.821
5	1.000	1.000	1.000	1.000
D_{max}	0.030	0.027	0.002	—

Table IV. Critical points chosen at each iteration.

Iteration	The $n+2$ points used (see Figure 3)									
1st	1	2	3	4	7	8	9	10	11	12
2nd	1	2	3	6	7	8	9	10	11	12
3rd	1	2	3	6	7	8	9	10	11	12

6. *Examples.* The procedure was tested on the Burroughs B5500 at the Stanford Computation Center using Burroughs Extended Algol.

We have chosen two examples to illustrate the use of the algorithm. The first is the function

$$f_1(x) = e^x \text{ on } [-1,1] \tag{6.1}$$

and the second is

$$\begin{aligned} f_2(x) &= 1 + x, && -1.0 \leq x < -0.5 \\ &= -x, && -0.5 \leq x < 0.0 \\ &= x, && 0.0 \leq x \leq 1.0. \end{aligned} \tag{6.2}$$

The first example, $f_1(x)$, is an infinitely differentiable function so that the error curve (4.1) is also differentiable, whereas $f_2(x)$ (see Figure 1) is continuous, but its derivative, $f_2'(x)$, has discontinuities at $x = -0.5$ and at $x = 0.0$, which cause the error curve to have a discontinuous derivative. We examine $f_2(x)$ as it provides an interesting example of approximating a function which is only continuous. In both cases we used Chebyshev polynomials as the Chebyshev system of functions.

Example 1. $[f_1(x) = e^x]$. Tables I and II show how the critical points and the coefficients of the approximating polynomial converge as we approximate $f_1(x) = e^x$ by a 4th-degree sum of Chebyshev polynomials. Figures differing from the final result are underlined at each step.

Table I shows that the coefficients of the "best" polynomial have converged to 6D after only one iteration; however, the critical points don't converge until the second iteration as shown by Table II. In other words, the polynomial does not change coefficients very much with a small change in the critical points. The starting points shown in Table II are chosen by *exchange* from $6 \times (n+2) = 36$ (for $n=4$) equally spaced points in the interval $[-1,1]$.

Various methods for choosing the starting values for the critical points have been proposed. These include the zeros of $T_{n+1}(x) - T_{n-1}(x)$, which are also the extrema of $T_{n+1}(x)$, and what we propose here is to let *exchange* choose $n + 2$ points from some original set of $k(n+2)$ points where $k \geq 1$. The original $k(n+2)$ points may be equally spaced, or they may be the zeros of $T_{k(n+2)+1}(x) - T_{k(n+2)-1}(x)$.

Table III compares various starting values for this example, $f_1(x) = e^x (n=4)$. D_{max} represents the maximum deviation from the "TRUE" values.

Example 2. $[f_2(x)]$. Approximation of $f_2(x)$ by an 8th degree sum of Chebyshev polynomials $(n=8)$ poses the problem of having an error curve with more than $N + 2$ local extrema. This problem also arises when approximating an even or odd function (see [6]). We resolve the problem by including all the local extrema of the error function, $\epsilon(x)$, which have the alternation of sign property, in the search for $n + 2$ critical points. That is, if the abcissas of the extrema are ordered algebraically, the signs of the corresponding ordinates must alternate. We obtain starting guesses for local extrema by having *exchange* pick $n + 2$ starting points from some original set of points, together with the corresponding first approximating polynomial, and then examining the resultant residuals. If the table of residuals indicates an extremum not already chosen by *exchange*, which has the correct alternating sign, then the corresponding abcissa is included as a critical point for later iterations. k must be chosen greater than 1 in order for this method to work.

Figure 2 shows the error curve, $\epsilon(x)$, for the first and third iterations of approximating $f_2(x)$ by an 8th-degree linear combination of Chebyshev polynomials.

Table IV indicates how the choice of critical points can change from one iteration to the next. If we had not included the additional extrema at points 5 and 6 at the first iteration, we would have arrived at the approximation whose error curve is illustrated by Figure 3. That is $n + 2$ extrema of the error curve have equal magnitude and alternating signs, but another extremum exists with larger modulus.

Fig. 2

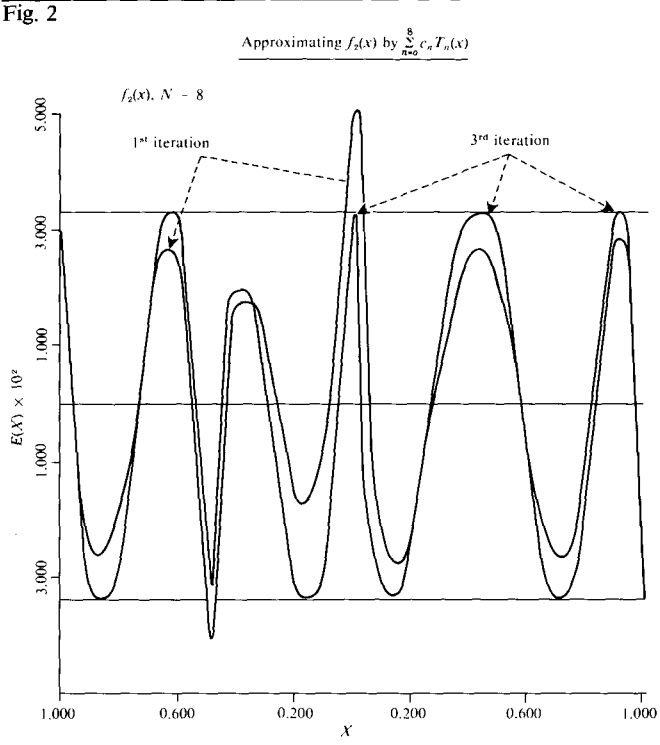

Fig. 3

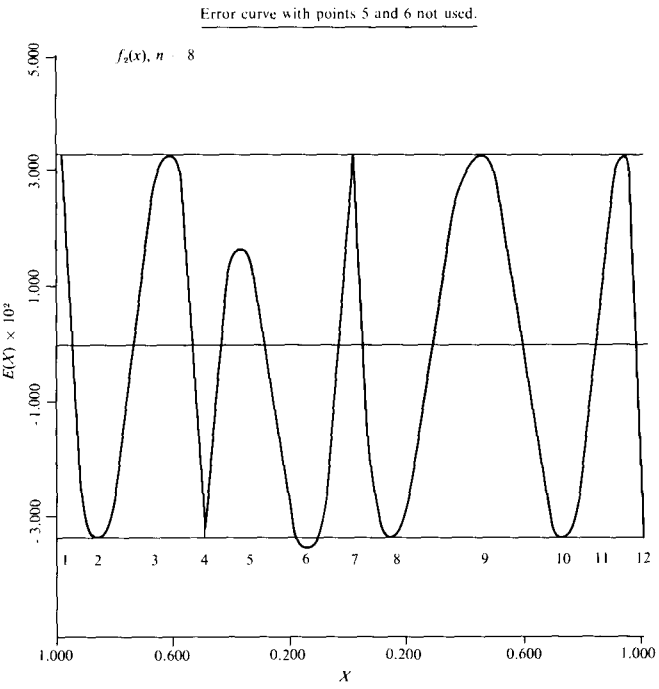

Table V. Comparison of starting values x_j for $f(x) = f_2(x)$, $n = 8$ (to 4D)

j	$T_9(x) - T_7(x) = 0$	*exchange* on 33 points equally spaced	*exchange* on 201 points equally spaced	TRUE (computed)
0	−1.0000	−1.0000	−1.00	−1.0000
1	−0.9397	−0.8750	−0.86	−0.8565
2	−0.7660	−0.6250	−0.62	−0.6248
3	−0.5000	−0.1250	−0.14	−0.1424
4	−0.1736	0.0	0.0	0.0
5	0.1736	0.1250	0.15	0.1456
6	0.5000	0.4375	0.44	0.4413
7	0.7660	0.7500	0.73	0.7290
8	0.9397	0.9375	0.93	0.9289
9	1.0000	1.0000	1.000	1.0000
D_{max}	0.3750	0.0210	0.0048	—

As an interesting comparison to Table III we give a similar table for $f(x) = f_2(x)$. D_{max} represents the maximum deviation from the "TRUE" values in Table V.

7. *Use of orthogonal polynomials.* Consider the polynomials $p_0(x), p_1(x), \cdots, p_n(x)$ orthogonal on the set of points $x_0 < x_1 < \cdots < x_m$. Such polynomials are described by Forsythe [13], and they form a Chebyshev system. This is easily seen since any licear combination,

$$P(x) = \sum_{i=0}^{n} c_i p_i(x), \tag{7.1}$$

is a polynomial of degree n which has exactly n zeros. Hence on any interval, $P(x)$ has no more than n zeros. This satisfies the definition of a Chebyshev system.

It is known, see Forsythe [13], that orthogonal polynomials have advantages over standard polynomials in least squares data-fitting. In the Remez algorithm, if a new set of polynomials, orthogonal on the critical points, is computed each time the critical points are adjusted, convergence is assured. This can be proved by nothing that at each iteration the best orthogonal polynomial fit is equivalent to the best fit that would be obtained if the Chebyshev system were held constant as standard polynomials. Perhaps this use of orthogonal polynomials will have computational advantages over, say, standard polynomials on the interval [0,1].

The use of orthogonal polynomials for the Chebyshev system has been implemented and tried successfully on a Burroughs B5500 but as yet we have no illustrations of any dramatic advantages over any other Chebyshev system.

References

1. Remez, E.Y. General computational methods of Chebyshev approximation. In *The Problems with Linear Real Parameters*, AEC-tr-4491, Books 1 and 2, English translation by US AEC.
2. Stiefel, E.L. Numerical methods of Chebyshev approximation. In *On Numerical Approximation*, R.E. Langer (Ed.) U. of Wisconsin Press, Madison, 1959.
3. Achieser, N.I. *Theory of Approximation.* (Trans. by C.J. Hyman), Frederick Ungar Publ. Co., New York, 1956.
4. Novodvorskii, E.N., and Pinsker, I.S. On a process of equalization of maxima. *Uspehi Mat. Nauk. 6* (1951), 174–181. (Trans. by A. Shenitzer, available from New York U. Library.)
5. Muller, D.E. A method for solving algebraic equations using an automatic computer. *Math Tables Aids Comp. 10* (1956), 208–215.
6. Murnaghan, E.D., and Wrench, J.W. Rep. No. 1175, David Taylor Model Basin, Md., 1960.
7. Lawson, C.L. Private communication.
8. Fraser, W. A survey of methods of computing minimax and near minimax polynomial approximations for functions of a single independent variable. *J. ACM 12* (July 1965), 295–314.
9. Rivlin, T.J., and Cheney, E.W. A comparison of uniform approximations on an interval and a finite subset thereof. *SIAM J. on Numer. Anal. 3* (June 1966).
10. Bartels, R.H., and Golub, G.H. Computational considerations regarding the calculation of Chebyshev solutions for overdetermined linear equation systems by the exchange method. Tech. Rep. No. CS67, Comput. Sci. Dep., Stanford U. (June 1967). Also Algorithm 328 *Comm. ACM 11* (June 1968), 401–406, 428–430.
11. Rice, J.R. *The Approximation of Functions*, Vol. 1, Reading Mass. Addison-Wesley, 1964.
12. Veidinger, L. On the numerical determination of the best approximations in the Chebyshev sense. *Numer. Math. 2* (1960), 95–105.
13. Forsythe, G.E. Generation and use of orthogonal polynomials for data-fitting with a digital computer. *J. SIAM 5* (June 1957), 74–88.

Algorithm

```
procedure remez (n, a, b, kstart, kmax, loops, f, chebyshev, eps,
  exchange, c, emax, trouble, why);
  value n, a, b, kstart, kmax, loops;
  real array c; real a, b, emax; label trouble;
  integer n, kstart, kmax, loops, why;
  real procedure f, eps; procedure chebyshev, exchange;
comment Procedure remez finds the best fit (in the minimax sense) to
  a function f using a linear combination of functions which form a
  Chebyshev system. The exchange algorithm of E.L. Stiefel is used
  to obtain starting values for the critical points and the Remez
  algorithm is then used to find the best fit;
begin
  procedure quadraticmax(n, x, niter, alfa, beta, ok, a, b, c, nadded,
    eps);
    value n, niter, alfa, beta, nadded; array x, c;
    integer n, niter, nadded; real alfa, beta, a, b;
    Boolean ok; real procedure eps;
  comment Procedure quadraticmax is called to adjust the values of
    the critical points in each iteration of the Remez algorithm. The
    points are adjusted by fitting a parabola to the error curve in a
    neighborhood, or if that proves unsatisfactory a brute force de-
    termination of the extrema is used;
  begin
    integer i, count1, count2, nhalf, signepsxstar, signu, signv, signw,
      jmax, ncrude, j, nn;
    real u, v, w, denom, epsu, epsv, epsw, xstar, epsxstar, xxx, misse,
      missx, dx, emax, etmp;
    integer array signepsx [0 : n + 1]; array epsx [0 : n + 1];
    nn := n − nadded;
    comment On arbitrary parameters...
      ncrude The number of divisions used in the brute force
        search for extrema.
      nhalf The parameter (alpha) which determines the size of
        interval to be examined for an extremum is reduced by
        half if a bad value for xstar is computed, however this
        reduction may occur only nhalf times.
      misse If the value of the error curve at a new critical point
        differs from the previous value by a relative difference of
        more than misse then the brute force method is brought in.
      missx The brute force method keeps searching until it is
        within missx of an extremum;
    comment Set values of the constants;
    ncrude := 10; nhalf := 4; misse := 1.0₁₀ −2; missx :=
      1.0₁₀ −5;
    comment Compare missx with absepsx. They should be equal;
    for i := 0 step 1 until n + 1 do
    begin
      epsx[i] := eps(x[i], c, nn);
      signepsx[i] := sign(epsx[i]);
    end;
    for i := step 1 until n + 1 do
```

```
    begin
      comment If the starting values for the critical points do not
        alternate the sign of eps(x), then we go to the label trouble;
      if signepsx[i] × signepsx[i−1] ≠ −1 then go to trouble;
    end;
    comment First find all the interior extrema. Then we will find
      the end extrema, which may occur at the ends of the interval;
    for i := 1 step 1 until n do
    begin
      count1 := 0;  count2 := 0;
L1:
      u := x[i];
      v := u + alfa × (x[i+1] − u);  w := u + alfa ×
        (x[i−1] − u);
      epsu := epsx[i];  signu := signepsx[i];
      epsv := eps(v, c, nn);  signv := sign(epsv);
      epsw := eps(w, c, nn);  signw := sign(epsw);
      if ¬ signu = signv ∨ ¬ signv = signw then go to L3;
      comment If the sign of eps(x) at the three points is not the
        same, we go to L3 where alfa is reduced to make the points
        closer together;
      epsu := abs(epsu);  epsv := abs(epsv);  epsw := abs(epsw);
L2:
      denom := 2.0 × ((epsv − epsu) × (w − u) + (epsw −
        epsu) × (u − v));
      if denom = 0.0 then xstar := 0.5 × (v + w) else xstar :=
        0.5 × (v + w) + (v − u) × (u − w) × (epsv − epsw)/
        denom;
      count1 := count1 + 1;
      comment Test xstar to be sure it is what we want. Is it be-
        tween x[i−1] and x[i+1]? Is eps(xstar) ≥ eps(u, v, w)? If
        xstar is too bad, go to L3 and reduce alfa unless alfa has
        been reduced nhalf times. Otherwise if ok, go to savexstar;
      if xstar = u ∨ xstar = v ∨ xstar = w then
      begin
        epsxstar := eps(xstar, c, nn); signepsxstar := sign
        (epsxstar);
        epsxstar := abs(epsxstar);  go to savexstar
      end;
      if xstar ≤ x[i−1] ∨ xstar ≥ x[i+1] then go to L3;
      epsxstar := eps(xstar, c, nn);
      signepsxstar := sign(epsxstar);
      epsxstar := abs(epsxstar);
      if signepsxstar ≠ signu ∨ epsxstar < epsu ∨ epsxstar <
        epsv ∨ epsxstar < epsw then
      begin
        if epsu ≥ epsv ∧ epsu ≥ epsw then
        begin
          if abs(epsxstar − epsu) > misse × epsu then go to
            LBL2;
          xstar := u;  epsxstar := epsu;  signepsxstar := signu·
          go to savexstar;
        end;
        if epsv ≥ epsu ∧ epsv ≥ epsw then
        begin
          if abs(epsxstar − epsv) > misse × epsv then go to
            LBL2;
          xstar := v;  epsxstar := epsv;  signepsxstar := signv;
          go to savexstar·
        end;
        if abs(epsxstar − epsw) > misse × epsw then go to
          LBL2;
        xstar := w;  epsxstar := epsw;  signepsxstar := signw;
        go to savexstar;
LBL2:
        jmax := 0;
LBL1:
        dx := (v−w)/ncrude;  emax := 0.0;  xxx := w − dx;
        for j := 0 step 1 until ncrude do
        begin
          xxx := xxx + dx;  jmax := jmax + 1;
          etmp := eps(xxx, c, nn);
          if abs(etmp) > emax then
          begin
            emax := epsxstar := abs(etmp);
            signepsxstar := sign(etmp);
            u := xstar := xxx;
            v := u + dx;  w = u − dx;
          end
        end;
        if dx > missx then go to LBL1;
        comment Make sure v and w are within bounds;
        if v ≥ x[i+1] then go to L3;
        if w ≤ x[i−1] then go to L3;
        go to savexstar
      end;
      if count1 > niter then go to savexstar;
      if epsu ≤ epsw then
      begin
        if epsv < epsu then
L4:
        begin
          comment v is minimum;
          if xstar > u then
          begin
            v := xstar;  epsv := epsxstar;  go to L2;
          end;
          if xstar > w then
          begin
            epsv := epsu;  v := u;
            epsu := epsxstar;  u := xstar;
            go to L2;
          end
          else
          begin
            v := u;  epsv := epsu;
            u := w;  epsu := epsw;
            w := xstar;  epsw := epsxstar;
            go to L2;
          end;
        end
        else
        begin
          comment u is minimum;
          if xstar ≥ v then
          begin
            u := v;  epsu := epsv;
            v := xstar; epsv := epsxstar;
            go to L2;
          end;
          if xstar ≥ w then
          begin
            u := xstar;  epsu := epsxstar;
            go to L2;
          end
          else
          begin
            u := w;  epsu := epsw;
            w := xstar; epsw := epsxstar;
            go to L2;
          end;
        end;
      end
      else
      begin
        if epsv < epsw then
        begin
          comment v is minimum;  go to L4;
        end
        else
        begin
          comment w is minimum;  if xstar ≥ v then
```

```
            begin
              w := u;  epsw := epsu;
              u := v;  epsu := epsv;
              v := xstar;  epsv := epsxstar;
              go to L2;
            end;
            if xstar ≥ u then
            begin
              w := u;  epsw := epsu;
              u := xstar; epsu := epsxstar;
            go to L2;
            end
            else
            begin
              w := xstar;  epsw := epsxstar;
              go to L2;
            end;
          end;
        end;
L3:
        count2 := count2 + 1;
        if count2 > nhalf then go to trouble;
        alfa := 0.5 × alfa;
        comment The factor 0.5 used in reducing alpha is arbitrarily
          chosen;
        go to L1;
        comment Replace x[i] by xstar after checking alternation of
          signs;
      savexstar:
        if i > 1 ∧ signepsxstar × signepsx[i−1] ≠ −1 then go to
          trouble;
        signepsx[i] := signepsxstar;
        x[i] := xstar;
      end;
      comment This is the end of the loop on i which finds all interior
        extrema. Now we proceed to locate the extrema at or near
        the two endpoints (left end, then right end);
      comment We assume beta > alfa;
      for i := 0, n + 1 do
      begin
        count1 := 0;  count2 := 0;
L8:
        u := x[i];  if i = 0 then
        begin
          if a < u then w := u + alfa × (a − u) else w := u +
          beta × (x[1] − u);
          v := u + alfa × (x[1] − u);
        end
        else
        begin
          if b > u then w := u + alfa × (b − u) else w := u +
          beta × (x[n] − u);
          v := u + alfa × (x[n] − u);
        end;
        epsu := epsx[i];  signu := signepsx[i];
        epsv := eps(v, c, nn);  signv := sign(epsv);
        epsw := eps(w, c, nn);  signw := sign(epsw);
        if signv ≠ signu ∨ signv ≠ signw then go to L7;
        epsu := abs(epsu);  epsv := abs(epsv);  epsw := abs(epsw);
L5:
        denom := 2.0 × (epsu × (v−w) + epsv × (w−u) + epsw ×
          (u−v));
        if denom = 0.0 then xstar := 0.5 × (w+v) else xstar :=
          0.5 × (v+w) + (v−u) × (u−w) × (epsv − epsw)/
          denom;
        if i = 0 ∧ (xstar < a ∨ xstar ≥ x[1]) then
        begin
          xstar := a;  epsxstar := eps(a, c, nn);
          signepsxstar := sign(epsxstar); epsxstar := abs (epsxstar);
        end
        else
        if i = n + 1 ∧ (xstar > b ∧ xstar ≤ x[n]) then
        begin
          xstar := b;  epsxstar := eps(b, c, nn);
          signepsxstar := sign(epsxstar); epsxstar := abs (epsxstar);
        end
        else
        begin
          epsxstar := eps(xstar, c, nn);
          signepsxstar := sign(epsxstar);
          epsxstar := abs(epsxstar);
        end;
        count1 := count1 + 1;
        if i = 0 ∧ xstar ≥ x[1] then go to L7;
        if i = n + 1 ∧ xstar ≤ x[n] then go to L7;
        if xstar = u ∨ xstar = v ∨ xstar = w then go to L6;
        if signepsxstar ≠ signu ∨ epsxstar < epsu ∨ epsxstar <
          epsv ∨ epsxstar < epsw then
        begin
          if epsu ≥ epsv ∧ epsu ≥ epsw then
          begin
            xstar := u;  epsxstar := epsu;
            signepsxstar := signu;  go to L6;
          end;
          if epsv ≥ epsu ∧ epsv ≥ epsw then
          begin
            xstar := v;  epsxstar := epsv;
            signepsxstar := signv;  go to L6;
          end;
          xstar := w;  epsxstar := epsw;
          signepsxstar := signw; go to L6;
        end;
        if count1 > niter then go to L6;
        if epsu < epsw then
        begin
          if epsv < epsu then
          begin
            comment v is minimum;
            v := xstar;  epsv := epsxstar;
            go to L5;
          end
          else
          begin
            comment u is minimum;
            u := xstar;  epsu := epsxstar;
            go to L5;
          end;
        end
        else
        begin
          if epsv < epsw then
          begin
            comment v is minimum;
            v := xstar;  epsv := epsxstar;
            go to L5;
          end
          else
          begin
            comment w is minimum; w := xstar; epsw := epsxstar;
            go to L5;
          end
        end;
L7:
        count2 := count2 + 1;
        if count2 > nhalf then go to trouble;
        alfa := 0.5 × alfa;  beta := 0.5 × beta;
        go to L8;
        comment Replace x[i] by xstar after checking its sign;
```

```
L6:
        if i = 0 ∧ signepsxstar × signepsx[1] ≠ − 1 then go to
          trouble;
        if i ≠ 0 ∧ signepsxstar × signepsx[n] ≠ − 1 then go to
          trouble;
        signepsx[i] := signepsxstar;  x[i] := xstar;
      end;
      go to done;
trouble:
      ok := false;  go to L9;
done:
      ok := true;
L9:
    end  quadraticmax;
    comment Procedure start computes the arrays which are then in-
      put to exchange to find the best approximation on the points at
      hand;
    procedure start (m, n, a, d, xi, chebyshev, f);
      value m, n;  integer m, n;
      array a, d, xi;
      procedure chebyshev;  real procedure f;
    begin
      integer i, j;  real array t[0:n];
      for i := 0 step 1 until m do
      begin
        chebyshev (n, xi[i], t);
        for j := 0 step 1 until n do a[i,j] := t[j];
        d[i] := f(xi[i]);
      end
    end start;
    comment Now the procedure remez;
    real epsc, alfa, beta, epsx, absepsc, absepsx, rcompare, dx, maxr,
      minr, tempr, minsep;
    integer m, i, itemp, j, niter, nloop, k, nadded, isub, maxri, ilast,
      signnow, jj;
    integer signnew;  integer array refset[0 : n + 1 + n];
    comment Assume number of points added ≤ n;
    integer array ptsadd[0 : n];
    array clast[0 : n + 1], xq, xqlast[0 : n + 1 + n];
    Boolean firsttime, ok, convx, convc, addit;
    why := 0; k := kstart;
    comment Come here if k gets changed;
newk:
    m := n + 1 +(k − 1) × (n + 2);
    begin
      array r, xi, d[0 : m], aa[0 : m, 0 : n + 1];
      firsttime := true;  convx := false; convc := false;
      nloop := 0;
      comment This makes the initial points spaced according to the
        extrema of the Chebyshev polynomial of degree m − 1;
      for i := 0 step 1 until m do
      xi[i] := (a+b)/2.0 − (b−a) × cos((3.14159265359 × i)/m)/
        2.0;
      comment 3.14159... is π;
      dx := (b−a)/m;
      comment To use equally spaced points a statement such as the
        following could be used. for i := 0 step 1 until m do xi[i] :=
        a + i × dx;
      start(m, n, aa, d, xi, chebyshev, f),
    comment The following constants are used in testing for conver-
        gence
        epsc used in relative test on coefficients
        absepsc used in absolute test on coefficients
        epsx used in relative test on critical points
        absepsx used in absolute test on critical points
        rcompare used to compare relative magnitudes of max and
        min values of residual on the critical points;
      epsc := 1.0₁₀ − 7; absepsc := 1.0₁₀ − 7;  epsx := 1.0₁₀ − e·
        absepsx := 1.0₁₀ − 5;
      rcompare := 1.0000005;
      comment epsx and absepsx should be the same as missx in pro-
        cedure quadraticmax. epsc and absepsc should be adjusted
        according to knowledge of the expected magnitudes of the
        coefficients (if known). It is best to depend on the critical
        points and/of the max and min of the residuals for conver-
        gence criteria;
      comment Now call on exchange to find the first approximation
        to the best approximating function;
      exchange (aa, d, c, m, n, refset, emax, singular, r);
      comment The subscripts of the points chosen are in array ref-
        set[0:n+1], the coefficients of the best approximating func-
        tion on the m points are in c[0:n], the residuals in r;
      comment The reference set, the coefficients at this step, and/or
        the residuals may be written at this point;
      for i := 0 step 1 until n do clast[i] := c[i];
      comment Now we are going to look for any extrema not given
        by the points chosen by exchange;
      comment Make sure critical points are algebraically ordered;
      for i := 0 step 1 until n do for j := i + 1 step 1 until n + 1 do
      begin
        if refset[j] < refset[i] then
        begin
          itemp := refset[j];  refset[j] := refset[i];
          refset[i] := itemp;
        end
      end;
      nadded := 0;  maxr := 0;  maxri := 0;  ilast := 0;
      signnow := sign(r[0]);
      for i := 0 step 1 until m + 1 do
      begin
        if i = m + 1 then go to LBL;
        if sign(r[i]) ≠ 0 ∧ sign(r[i]) = signnow then
        begin
          if abs(r[i]) > maxr then
          begin maxri := i;  maxr := abs(r[i]);  end
        end
        else
LBL:
        begin
          if i < m + 1 then signnow := sign(r[i]);
          addit := true;
          for j := 0 step 1 until n + 1 do
          begin
            for jj := ilast step 1 until i − 1 do
            begin
              if jj = refset[j] then addit := false;
            end
          end;
          if addit then
          begin
            nadded := nadded + 1;  if nadded > n then
            begin
              comment We assume nadded is always ≤ n. If nadded
                is > n, why is set to −1 and we go to the label
                trouble. This can be modified by changing this test
                and changing the declarations for ptsadd, refset, xq,
                and xqlast above;
              why := −1;
              go to trouble
            end;
            ptsadd[nadded] := maxri;
            refset [n + 1 + nadded] := maxri;
          end;
          if i < m + 1 then
          begin
            ilast := i;  maxr := abs(r[i]);  maxri := i;
          end
        end
      end;
      comment We now have n + 2 + nadded points to send to
        quadraticmax for adjustment;
```

```
    m := n + nadded;
    comment Make sure critical points are algebraically ordered;
    for i := 0 step 1 until m do for j := i + 1 step 1 until m + 1
      do
    begin
      if refset[j] < refset[i] then
      begin
        itemp := refset[j]; refset[j] := refset[i];
        refset[i] := itemp;
      end
    end;
    for i := 0 step 1 until m + 1 do xq[i] := xi[refset [i]];
    niter := 2;
    comment This is the number of times to iterate in quadraticmax;
    alfa := 0.15; beta := 0.2;
    comment alfa and beta are used to determine the points used
      in quadraticmax to fit a parabola. They are arbitrary subject
      to: 0 < alfa < beta < 1. Also beta should be fairly small
      to keep the points on one side of zero;
    comment This is the beginning of the loop that calls on
      quadraticmax, exchange, etc.;
loop:
    nloop := nloop + 1;
    quadraticmax(m, xq, niter, alfa, beta, ok, a, b, c, nadded, eps);
    if ¬ ok then
    begin
      k := k + 1; if k > kmax then
      begin why := 1; go to trouble; end;
      go to newk;
    end;
    if ¬ firsttime then
    begin
      comment Compare the largest and smallest of the residuals
        at the critical points (after adjustment);
      comment Set minr to a large number;
      maxr := 0.0; minr := 1.0₁₀50;
      for i := 0 step 1 until n + 1 do
      begin
        addit := true;
        for j := 1 step 1 until nadded do if refset[i] = ptsadd[j]
          then addit := false;
        if addit then
        begin
          tempr := abs(eps (xq [refset [i]], c, n));
          if tempr > maxr then maxr := tempr else if tempr <
            minr then minr := tempr;
        end
      end;
      if maxr ≤ rcompare × minr then why := 4;
    end;
    comment Compare xq to xqlast;
    if ¬ firsttime then
    begin
      convx := true;
      for i := 0 step 1 until m + 1 do
      begin
        if abs(xq [i] − xqlast[i]) > absepsx then
        begin
          if abs (xq [i] − xqlast[i]) ≥ epsx × abs(xq [i]) ∧
            xq[i] ≠ 0.0 then convx := false;
          if xq[i] = 0.0 ∧ abs(xq [i] − xqlast[i]) > absepsx
            then convx := false;
        end;
        xqlast[i] := xq[i];
      end
    end
    else
    begin
      firsttime := false;
      for i := 0 step 1 until m + 1 do xqlast[i] := xq[i];
      for i := 0 step 1 until n do clast[i] := c[i];
    end;
    comment Get ready to call exchange again;
    start(m + 1, n, aa, d, xq, chebyshev, f);
    exchange(aa, d, c, m + 1, n, refset, emax, singular, r);
    comment Now compare the new coefficients to the last set of
      coefficients;
    if ¬ firsttime then
    begin
      convc := true;
      for i := 0 step 1 until n do
      begin
        if abs(c[i] − clast[i]) ≥ epsc × abs(c[i]) ∧ c[i] ≠ 0.0
          then convc := false;
        if c[i] = 0.0 ∧ abs(c[i] − clast[i]) > absepsc then
          convc := false; clast[i] := c[i];
      end
    end;
    comment Set the parameter why to the proper value according
      to the following:
        why = 4 if maxr ≤ rcompare × minr.
        why = 5 if "4" and convx = true.
        why = 6 if "4" and convc = true.
        why = 7 if "4" and convx = convc = true.
        why = 8 if convx = true.
        why = 9 if convc = true.
        why = 10 if convx = convc = true. Any value of why ≥
          4 indicates convergence;
    if why = 4 ∧ convx then why := 5;
    if why = 4 ∧ convc then why := 6;
    if why = 5 ∧ convc then why := 7;
    if why = 0 ∧ convx then why := 8;
    if why = 0 ∧ convc then why := 9;
    if why = 8 ∧ convc then why := 10;
    if why ≥ 4 then go to converged;
    if nloop ≥ loops then
    begin why := 3; go to trouble end;
    comment We go to label trouble in calling program if no con-
      vergence after a number of iterations equal to loops;
    go to loop;
singular:
    why := 2; go to trouble;
    comment We come to singular if exchange gets into trouble;
converged:
  end;
  comment End of block using m in array declarations;
  comment There are four exits to the label trouble . . .
    (why=1) if k gets > kmax
    (why=2) if exchange gets into trouble
    (why=3) if no convergence after iterating loops number of
      times
    (why=−1) if number of added points is greater than n;
end remez
```