1975

Algorithm - Adapt, Adaptive Smooth Curve Fitting

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
75-166

ALGORITHM - ADAPT, Adaptive Smooth Curve Fitting

John R. Rice

Mathematical Sciences 428

Purdue University
West Lafayette, Indiana 47907

October 1, 1975

CSD-TR 166

Abstract

This paper describes a fast and reliable algorithm which computes smooth piecewise polynomial approximations to functions. It adaptively locates the knots by a procedure that has been shown to provide the optimal rate of convergence as the accuracy requirements (and number of knots) increase. Local Hermite interpolation is used which requires that derivatives of the function be known (or estimated accurately) and that the polynomial degree plus one be twice the smoothness. The theoretical background and interesting algorithm components are described briefly then algorithm usage (user interface, role of two unusual arguments, portability) are discussed in more detail. A very brief summary is given of the extensive testing performed. The algorithm is first described in a very high level language (about 120 lines) and then given in Fortran (about 1250 lines). A test driver and a FUNCTION subprogram containing 20 functions (with derivatives) are also given which are useful for testing this and other similar algorithms.

Keywords: piecewise-polynomial approximation, adaptive curve fitting, adaptive approximation, Hermite interpolation

CR Category: 5.13

ALGORITHM - ADAPT, Adaptive Smooth Curve Fitting

John R. Rice*

1. Introduction and Background. The basic objectives of this approximation algorithm are: speed, reliability and smoothness. Algorithms already exist with any two of these three properties. Speed requires that the work be proportional to the length of the curve (for fixed accuracy and more or less uniform complexity of the function). Reliability requires that curves with singularities or near singularities, oscillations and other complex behavior be handled. Smoothness (number of continuous derivatives) of the approximation obtained is input to the algorithm. The author believes that adaptive piecewise polynomial algorithms offer the best hope for such algorithms with nonadaptive piecewise polynomial schemes or rational function approximation as the only serious competitors. Non-adaptive schemes do not cope efficiently with functions having very non-uniform behavior (singularities in slopes for example) and the work for rational approximation probably increases faster than linearly with the length of the curve.

The theoretical background for this algorithm is provided by Rice [1] and the references cited there. This may be summarized by saying that for given fixed degree n of the pieces then the error of the best piecewise polynomial approximation behaves like k^{-n} where k is the number of pieces. This theoretical result applies to a broad class of functions which includes everything that conceivably could arise in applications. A class of adaptive algorithms has been found to choose the knots so that the error behaves like k^{-n} even though the best approximation is not obtained. These algorithms apply to any piecewise smooth function with a finite number of "algebraic" singularities, i.e. the function $f(x)$ behaves like $a + b(x - s)^{\alpha}$ near the singularity s. The exponent α must be so that the norm of $f(x)$ is finite i.e. $\alpha > -1/2$ for least squares, $\alpha > 0$ for uniform approximation.

---

*This work was supported in part by grant GP 32940X from the National Science Foundation.

The key ingredients in these algorithms are a local approximation operator, a local error estimator and a data structure for the subintervals generated. ADAPT uses Hermite interpolation at subinterval end points plus ordinary interpolation in between if needed. Error estimates are made by a simple Gauss quadrature formula and a stack is used for the intervals. This is probably the simplest choice from the set of algorithms currently known to give the optimal convergence rate. A more detailed description of ADAPT is given in Section 5. Note that the nature of ADAPT requires that f(x) and its derivatives be available for arbitrary x (see [2] for guidance on numerically estimating derivatives) and thus ADAPT is not directly applicable to discrete data sets.

The next section presents some remarks about the algorithm components; most of them incorporate rather standard methods. Section 3 discusses the use of ADAPT (input/output, role of the arguments CHARF = characteristic length of f(x) and EDIST = error distribution type, and portability). The fourth section describes the extensive testing performed and summarizes the algorithm properties observed in [2]. It also briefly describes the driver program and 20 functions with derivatives used for testing ADAPT and which are applicable to similar algorithms. The final section has a very high level description of the algorithm and ADAPT itself is available through the ACM Algorithm Distribution Service.

2. The Principal Algorithm Components.

The algorithm ADAPT is to approximate the function F(x) on the interval [A,B] to within an accuracy ACCUR and by piecewise polynomials of degree DEGREE with SMOOTH continuous derivatives. The error is measured in the L_p-norm with p set by NORM. Other input is the characteristic length CHARF of F, printed output level LEVEL, the error distribution type EDIST and the number NBREAK of break points plus information about the break points if NBREAK > 0. The output is the array XKNOTS of knots, the coefficients COEFS (relative to the knot locations) of the polynomial pieces, the estimated error ERROR and number KNOTS of knots. The approximation obtained is automatically available as the FUNCTION subprogram PPOLY.

2.1 Data structure and discard procedure for subintervals. ADAPT generates a set of subintervals of [A,B] which are maintained in a stack with the leftmost on the top. Since subintervals are created by halving, the maximum size of the stack is limited by the machine word length.

Intervals are discarded whenever the estimated error on an interval is small enough. This decision made in the subprogram CHECK is somewhat subtle and three strategies are provided. This question is discussed in some detail later in connection with the argument EDIST which indicates the strategy selected by the user. Two subprograms, PUT and TAKE, are used for access to the interval stack.

2.2 Hermite interpolation. The values of DEGREE and SMOOTH are specified by the user and DEGREE $\geq$ 2 * SMOOTH + 1. The polynomial pieces are determined by interpolating SMOOTH derivatives of F, at each end point of a subinterval plus the value of F, at DEGREE - 2 + SMOOTH - 1 other points. During the computation the polynomial pieces are represented by divided differences computed in NEWTON and used by POLYDD. Once a polynomial piece is accepted for the final approximation PTRANS transforms its representation into powers with origin at the left end point of the subinterval. The transformation is accomplished by repeated synthetic division. The subprogram COMPUT controls the computation of a polynomial piece as well as the error estimation.

2.3 The break point mechanism. It is sometimes very useful for a user to be able to specify breaks in some derivative at certain points. Most commonly one has a known or desired jump in the slope at a given point and ADAPT allows this via NBREAK and associated arguments XBREAK, DBREAK, BLEFT, BRIGHT which specify the exact nature of the break point. This rather straightforward facility is implemented primarily in TAKE with some impact on COMPUT.

2.4 Error estimation. The L_p-norm of the local error is estimated in ERRINT by a 4-point Gauss quadrature for (F - POLYDD)**P on the subinterval under consideration. Special code is used for P=infinity, minimax approximation. The global error estimate is built up in PUT by appropriately combining the local error estimates.

2.5 Fatal errors. The algorithm normally terminates when the stack is empty. The stack should not overflow but might do so at very strong singularities which cause the algorithm to want to operate at accuracies inconsistant with the machine word length. This overflow has not occured in the testing so far, but if it does a message such as the following is printed:

INTERVAL DIVIDED TOO MUCH, EXCEEDED LIMIT 50 ON INTERVAL STACK AT

17923.12345678 17923.12345687

INTERVAL DISCARDED AND COMPUTATION CONTINUED

This message may be suppressed by setting LEVEL = -1 and note that the computation is allowed to proceed on the conjecture that this situation is not truly fatal.

A strong singularity has been observed to cause another situation indicated by a message like

GOT SHORT INTERVAL **** 3210.12345678 3210.12345679 **** DISCARD IT

which may also be suppressed by setting LEVEL = -1. Experiments indicate that the algorithm will recover and produce satisfactory results provided it can discard (and ignore) enough short intervals to get out of the region where the machine word length is inadequate. It often cannot get out of this region before exceeding run time limits or, more likely, the limit on the number of knots. The variable BUFFER in PUT governs short interval detection.

The arrays XKNOTS and COEFS are passed to ADAPT with variable dimensions KDIMEN and NDIMEN. If the number of knots computed exceeds KDIMEN then a fatal error message is printed and the computation aborted. This message cannot be suppressed. Checks are made on various input parameters by SETUP and inconsistant or impossible input leads to fatal error messages and a RETURN without any computation.

3. Algorithm Usage.

3.1 User interface. The input to ADAPT is via the COMMON block INPUT2 except for the function F. This is the most convenient for extensive use of ADAPT but not for occasional use. A subroutine PPFIT4 is provided which has all input as arguments. Entry points PPFIT1, PPFIT2 and PPFIT3 in PPFIT4 have fewer (10, 12, 15, respectively) arguments

than PPFIT4 (21 arguments). Due to the variable nature of entry point implementations, these are only indicated by COMMENT cards and local modifications are needed to activate these features.

The basic output is the arrays XKNOTS and COEFS which are arguments to ADAPT (and also the PPFIT subroutines) and the numbers KNOTS and ERROR in the COMMON block RESULZ. The PPFIT routines have KNOTS and ERROR as arguments. In addition the FUNCTION subprogram PPOLY (T, XKNOTS, COEFS, KDIMEN, NDIMEN) returns the value at the point T of the most recently computed approximation. It is automatically available to the user.

There are five levels (-1, 0, 1, 2, 3, 4) of printed output available. LEVEL = -1 only provides fatal error messages; LEVEL = 0 also provides "semi-fatal" error messages plus 1 line of output; LEVEL = 1 provides a print out of the input and the approximation obtained; LEVEL = 2 provides a condensed trace of the computation and the last two levels are only useful for debugging program modifications.

3.2 The characteristic length of F, CHARF. The correct operation of ADAPT depends on certain estimates being accurate which, in turn, depend on the relevant subintervals being small enough. Specifically, the sampling that ADAPT does of F must reveal the nature of F and not allow any significant features to go undetected. The nature of ADAPT is such that for half-decent functions it is very unlikely to miss significant features without using CHARF at all. However, knowing how ADAPT works, one can readily construct examples where it will fail unless CHARF is set properly. The value of CHARF is an upper limit on the size of the subintervals for polynomial pieces and it is to be set so that the 4-point Gauss quadrature formulas are reasonably (but not highly) accurate. This means that if F has some complex behavior on a very short segment, then setting CHARF to, say, half the length of this segment will force ADAPT to detect this behavior.

The argument CHARF is essential to proving ADAPT correct (which has not been attempted) but its practical value is debatable. If F is more or less uniformly complicated then short intervals are needed everywhere. It is extremely unlikely, but not impossible, that ADAPT would be fooled in such a case. If F is very smooth except on a very short segment, then ADAPT may well be fooled and setting CHARF small will avoid this. However, it will also force very small intervals where F is smooth and where they are not needed. Thus high reliability is obtained here at the cost of great inefficiency.

The nature of the approximations computed by ADAPT are such that there is a simple and efficient alternative to setting CHARF small. Let [C,D] be the subinterval of [A,B] where F is rough and let F be smooth elsewhere. One can then approximate F on [A,C], [C,D] and [D,B] independently and the approximations fit together smoothly at C and D to give a single smooth approximation for the entire interval [A,B].

3.3 The error distribution type, EDIST. The least squares error for the approximation S(x) to F(x) is

$$E(A,B) = [\int_A^B (F(x) - S(x))^2 dx]^{1/2}$$

To make $E(A,B) < .01$ is equivalent to making $E(A,B)^2 < .0001$ and, for any L_p-norm the program actually operates with $ACCUR^p$. Suppose A=0, B=1, then we can achieve $E(0,1)^2 < .0001$ by achieving $E(0,1/2)^2 < .0001/2$ and $E(1/2,1)^2 < .0001/2$ since $E(A,B)^2$ is simply additive over intervals. This approach is called proportional error distribution, the total error requirement is distributed over the subintervals of [A,B] in proportion to the lengths of the subintervals. This choice is selected by EDIST=0 and automatically results in the total error $E(A,B)^2$ less than the specified error $ACCUR^2$.

An alternative approach is to make the errors approximately equal on each of the subintervals independent of their lengths. This is called fixed error distribution and is selected by EDIST=2. The argument ACCUR is used for each subinterval and if k subintervals are finally used we see that the total error is then

$$[k * ACCUR^p]^{1/p} = \sqrt[p]{k}\ ACCUR$$

and thus ACCUR is not the specified total error when EDIST=2. This alternative is awkward to use because the final approximation error depends on the number of subintervals required which, of course, is unknown until after the approximation is computed. However, for rough or singular F(x) this disadvantage is more than compensated by the superior performance of this error distribution type. This is seen in the theory and verified in actual use.

A compromise approach called approximate fixed error distribution is selected by EDIST=1. Basically the algorithm keeps a running estimate of the final number of intervals it will use and adjusts the error requirement for subintervals accordingly. This approach is obviously not fool proof, but the testing reported below shows it to be 98 to 99% reliable. In any case, the total error actually obtained is available for the user to see and test.

We observe that for p = infinity (NORM=3) there is no difference between the fixed and proportional error distributions. For smooth, uniformly varying, F(x) the proportional error distribution gives perfectly satisfactory efficiency.

3.4 Portability considerations. Considerable pains have been taken to make the Fortran program portable. It is written in a subset of ANSI Fortran specified by PFORT (see [5]) except that four characters are packed per word rather than one as specified by PFORT. The current version is in single precision and specifically tailored to a machine with a long word length (CDC 6000-7000 series computers). Specific directions are given in the comments for changing the precision or to use it on machines with a shorter word length. In particular, all REAL variables are explicitly declared to facilitate the change to DOUBLE PRECISION. The program has been generated by an experimental Fortran converter which automatically produces versions tailored for different machines and precisions. Several of these versions have been produced and run successfully.

4. Algorithm Testing and Verification. This algorithm is based on a method with theoretically known properties. Care was taken to adhere to the requirements of that theory and considerable analysis has been made of various features and parts of the algorithm. However, no formal proof has been attempted since, as with many numerical algorithms, one cannot say in any a priori way what is to be computed. A somewhat related algorithm has been proved correct with a few idealizing assumptions (infinite precision arithmetic, infinite memory for example), see [3], [4].

Very extensive testing of the algorithm has been performed to see if the theoretical expectations are, in fact, realized by this algorithm. There are some approximation methods where this has not been the case. These tests are discussed at some length in [2] and we summarize them by saying that this algorithm performs as expected from the theory. The results in [2] give many insights into the practical use of this algorithm. About 2500 to 3000 different approximations have been computed in these tests and all have been examined for signs of incorrect performance. A few hundred of these runs were specifically designed to test the validity and correctness of the program.

The code with the ADAPT includes a set of 20 test functions (10 of them parameterized in various ways) and a driver to exercise ADAPT and measure its performance in various ways. These are included because it is felt they will be useful for testing other curve fitting programs. We note that the 20 test functions are chosen to present various features of the approximation problem and we found it amazingly difficult to get the first few derivatives of these functions computed correctly.

We give sample output for two functions, one easy and one difficult. The first is $F(x) = 1./(1+(x-2.5)^4)$ and LEVEL=2 output is given. The second function is shown in Figure 1 and LEVEL=0 output is given. In the first example the first and last three lines of output one from the test driver and the specific information about the problem is printed by ADAPT. For the second example the

approximation in on [2,15] for polynomial degree 6 with 1 continuous derivative; the requested and estimated accuracies are .001 and .00057 in the minimax norm; proportional error distribution was used with CHARF=4. The output has been reformatted to accomodate the narrowed page width here.

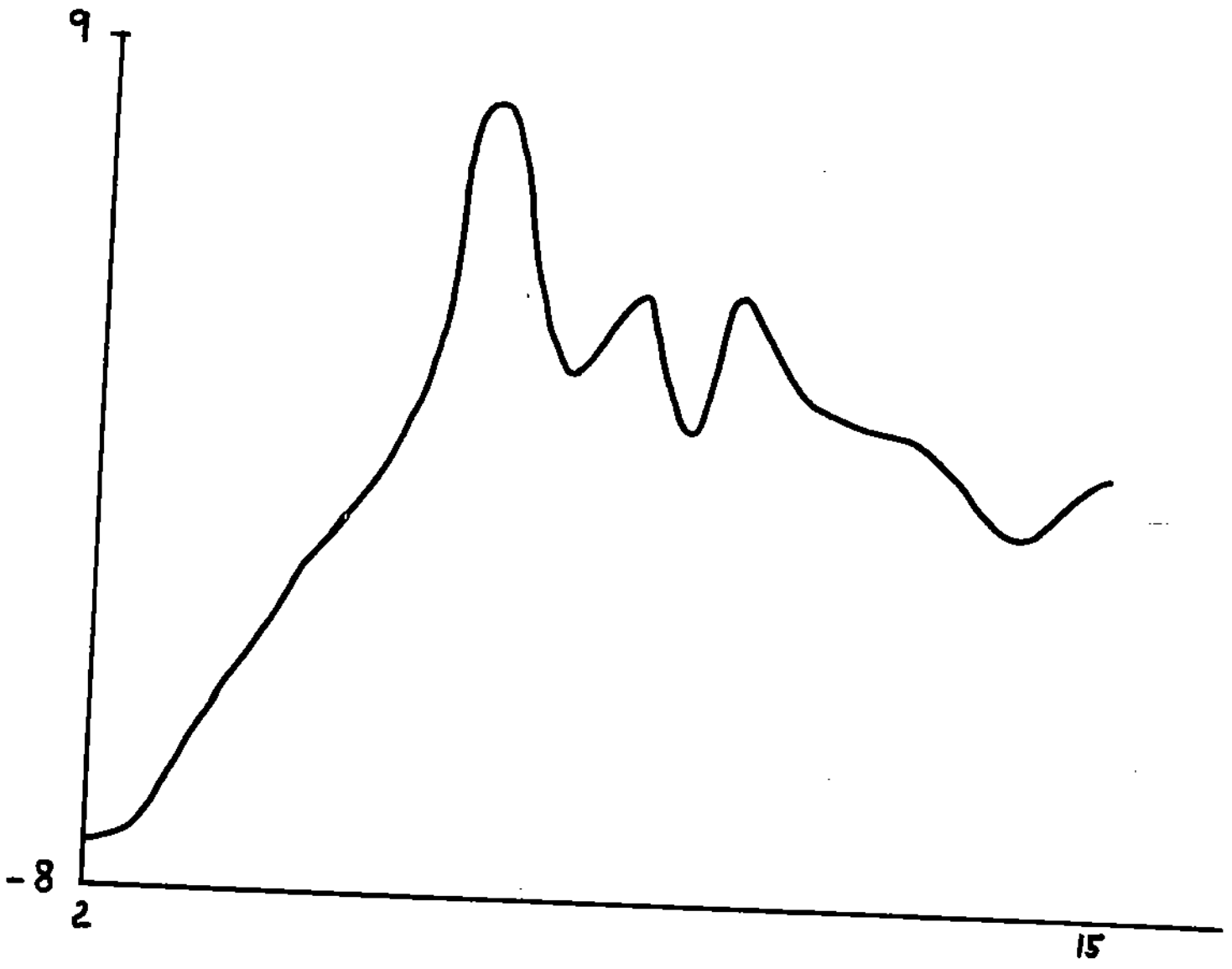

Figure 1. The complicated function with seven pieces approximated in the second example. The parameter P8F is 1.6 here rather than .6 as given in the DATA for F(x).

```
*****ADAPT FOR FUNCTION 4 = HUMP AT 2.5 RECIPROCAL OF QUARTIC
PIECEWISE POLYNOMIAL APPROXIMATION ON INTERVAL 1.0000E+00  5.0000E+00
OF DEGREE 5 WITH 2 CONTINUOUS DERIVATIVES
ACCURACY REQUESTED IS    1.0000E-03 MEASURED BY LEAST SQUARES
OTHER INPUT/DEFAULT VARIABLES ARE FOSCIL = 4.0000E+00
EMEAS =  2.0000E+00 ---- PROPORTIONAL ERROR DISTRIBUTION
KNOT  2 AT 1.50000E+00, LOCAL-GLOBAL ERRORS = 2.6008E-09  5.0998E-05
KNOT  3 AT 2.00000E+00, LOCAL-GLOBAL ERRORS = 3.5626E-08  1.9552E-04
KNOT  4 AT 2.50000E+00, LOCAL-GLOBAL ERRORS = 2.5115E-08  2.5168E-04
KNOT  5 AT 3.00000E+00, LOCAL-GLOBAL ERRORS = 2.5115E-08  2.9742E-04
KNOT  6 AT 3.50000E+00, LOCAL-GLOBAL ERRORS = 3.5626E-08  3.5225E-04
KNOT  7 AT 4.00000E+00, LOCAL-GLOBAL ERRORS = 2.6008E-09  3.5593E-04
KNOT  8 AT 5.00000E+00, LOCAL-GLOBAL ERRORS = 2.4834E-09  3.5940E-04

--- ADAPTIVE PIECEWISE POLYNOMIAL APPROXIMATION OF DEGREE 5 WITH
2 CONTINUOUS DERIVATIVES NEEDED 8 KNOTS FOR ERROR =       3.5940E-04
KNOT LOCATION          X-POWER COEFFICIENTS RELATIVE TO KNOT LOCATIONS
1   1.000000000000E+00  1.649484536082E-01
                        3.673078966947E-01
                        4.506148423367E-01
                        2.850451366482E-01
                        5.027123624777E-01
                       -9.058008728205E-01
2   1.500000000000E+00  5.000000000000E-01
                        1.000000000000E+00
                        5.000000000000E-01
                       -1.032973743131E+00
                       -2.117239975575E+00
                        2.484021982495E+00
3   2.000000000000E+00  9.411764705882E-01
                        4.429065743945E-01
                       -1.120293099939E+00
                        7.978831671079E-01
                        7.848565031541E-01
                       -1.003053124363E+00
4   2.500000000000E+00  1.000000000000E+00
                                         0
                                         0
                        1.400366374921E-01
                       -1.722776307754E+00
                        1.003053124363E+00
5   3.000000000000E+00  9.411764705882E-01
                       -4.429065743945E-01
                       -1.120293099939E+00
                       -9.426012619580E-01
                        4.092814980663E+00
                       -2.484021982495E+00
6   3.500000000000E+00  5.000000000000E-01
                       -1.000000000000E+00
                        5.000000000000E-01
                        9.740323204476E-01
                       -1.761789819574E+00
                        9.058008728205E-01
```

```
 7   4.000000000000E+00   1.649484536082E-01
                         -3.673078966947E-01
                          4.506148423367E-01
                         -3.547233211608E-01
                          1.658371414434E-01
                         -3.440822109296E-02

ADAPT USED 117 FUNCTION VALUES FOR ERRORS
SPECIFIED =  1.00000E-03 ESTIMATED BY ADAPT =  3.59399E-04
INDEPENDENT CHECK =  3.05041E-04 TIME USED  = .17200 SECONDS


*****ADAPT FOR FUNCTION 18 = COMPLICATED FUNCTION WITH 7 PIECES
---  ADAPTIVE PIECEWISE POLYNOMIAL APPROXIMATION OF DEGREE 6 WITH
1 CONTINUOUS DERIVATIVES NEEDED 30 KNOTS FOR ERROR = 5.7301E-04
```

5. High Level Expression of the algorithm ADAPT.

The following description of ADAPT indicates its basic structure and methods of computation.

```
PROCEDURE PPFIT - WITH ARGUMENTS =
    F       . FUNCTION TO FIT          LEVEL   . OUTPUT LEVEL -1 TO 2
    A,B     . INTERVAL ENDPTS          CHARF   . LIMIT ON PIECES LENGTH
    ACCUR   . ACCURACY DESIRED         EDIST   . TYPE OF ERROR CONTROL
    DEGREE  . POLYNOMIAL DEGREE        NBREAK  . NUMBER OF SPECIFIED
    SMOOTH  . NO. CONT. DERIVS                   BREAK POINTS IN FIT.
    NORM    . MEAS. OF L-P ERROR                 HAS RELATED ARGUMENTS
              P IN (0, INFINITY)                 GIVING DETAILS.

    OUTPUT = KNOTS   .                           . NUMBER OF KNOTS
             ERROR   .                           . ERROR ACHIEVED
             XKNOTS(K), K=1  TO KNOTS            . KNOT LOCATIONS
             COEFS(K,N),K=1  TO KNOTS            . POWER COEF. RELATIVE
                        N=1  TO DEGREE+1         . TO THE KNOT LOCATIONS

*** THIS PROGRAM MERELY COMPUTES A FEW DEFAULT VALUES
    AND PUTS VARIABLES IN COMMON BLOCKS, ETC.

    CALL ADAPT - TO DO THE APPROXIMATION
END PPFIT

SUBPROGRAM ADAPT
    CALL SETUP - CHECK INPUT, INITIALIZE THINGS, PRINT PROBLEM

    ***  LOOP OVER PROCESSING INTERVALS ***
```

```
      CALL TAKE    - AN INTERVAL OFF THE STACK
      CALL COMPUT  - AN APPROX ON THIS INTERVAL
      CALL CHECK   - FOR DISCARDING OR DIVIDING INTERVAL
      CALL PUT     - NEW INTERVALS ON STACK, UPDATE ALGORITHM STATUS
      CALL TERMIN  - TEST FOR FINISH, PRINT INTERMEDIATE OUTPUT
      IF NOT FINISHED - REPEAT LOOP

    CALL SUMMARY - FOR FINAL OUTPUT
END ADAPT

SUBPROGRAM SETUP
    SET LIMITS ON COMPUTATION PARAMETERS
    CHECK ALL INPUT DATA
    INITIALIZE VARIABLES AND INTERVAL STACK
    PRINT PROBLEM STATEMENT
END SETUP

SUBPROGRAM TAKE
    CHECK FOR BREAK POINT IN TOP INTERVAL
        IF SO - ADJUST XKNOTS TO MAKE IT A PARTITION POINT
        ELSE  - DO NOTHING
END TAKE

SUBPROGRAM PUT

    CHECK FOR DISCARDING INTERVAL
        IF SO - UPDATE ERROR ESTIMATE
                ADJUST STACK
                CALL PTRANS - TO OBTAIN COEFS FOR THIS INTERVAL
                UPDATE XKNOTS AND COEFS

        ELSE  - SUBDIVIDE INTERVAL AND PLACE 2 NEW ONES ON STACK
                CHECK FOR EXCEEDING MAX STACK SIZE OR OBTAINING
                      AN INTERVAL WHICH IS TOO SHORT.  SUCH SHORT
                      INTERVALS ARE DISCARDED WITHOUT REGARD TO
                      ERROR CONTROL POLICY AND WITH MESSAGE
END PUT

    SUBPROGRAM PTRANS - OF PUT
        CHANGES POLYNOMIAL REPRESENTATION FROM NEWTON DIVIDED
        DIFFERENCE FORM TO POWER FORM WITH ORIGIN SHIFTED TO THE
        XKNOT VALUE ON LEFT OF INTERVAL.  USES SYNTHETIC DIVISION
    END PTRANS

SUBPROGRAM COMPUT
    OBTAIN       - VALUES OF F AND DERIVATIVES.  MAKE ADJUSTMENTS
                   IF A BREAK POINT IS INVOLVED
    CALL NEWTON  - FOR DIVIDED DIFFERENCES OF INTERPOLATING
                   POLYNOMIAL FOR THIS INTERVAL
    CALL ERRINT  - TO ESTIMATE LOCAL ERROR + (F(X)-POLYDD(X))**P
                   FOR L-P NORM, 0 LT P LE INFINITY
```

```
END COMPUT

    SUBPROGRAM NEWTON - OF COMPUT
        BUILD UP TRUE DIVIDED DIFFERENCE TABLE WITH MULTIPLE
        POINTS AT THE INTERVAL ENDS PLUS INTERPOLATION POINTS
    END NEWTON

    SUBFUNCTION POLYDD - OF COMPUT
        EVALUATES POLYNOMIAL FROM DIFFERENCE TABLE OUT OF NEWTON
                  FOR L-P NORM WITH P IN (0, INFINITY)
    END POLYDD

    SUBPROGRAM ERRINT - OF COMPUT
        USES 4-POINT GAUSS QUADRATURE TO ESTIMATE ERROR NORM ON
        INTERVAL.  SPECIAL COMPUTATION FOR MAX-NORM, P=INFINITY.
    END ERRINT

SUBPROGRAM CHECK
    USES ERROR DISTRIBUTION TYPE AND CHARF TO DECIDE ON DISCARD
END CHECK

SUBPROGRAM TERMIN
    PRINT - INTERMEDIATE OUTPUT, IF ANY REQUESTED BY LEVEL = 2
    TEST  - FOR TERMINATION   EMPTY STACK - NORMAL
                              EXCEEDED XKNOTS LIMIT - ABNORMAL
END TERMIN

SUBPROGRAM SUMMARY
    LEVEL = -1    NOTHING
          =  0    1 LINE (OUTPUT RETURNED IN COMMON, ARGUMENTS)
          =  1    KNOTS AND COEFFICIENTS
          =  2          DITTO
END SUMMARY

FUNCTION PPOLY
    THE PIECEWISE POLYNOMIAL COMPUTED BY PREVIOUS CALL ON PPFIT
END PPOLY
```

6. References

1. Rice, John R., Adaptive approximation, J. Approx. Thy., to appear.

2. _____, On adaptive piecewise polynomial approximation, in Theory of Approximation (A.G. Law and B.N. Sahney, ed.) Academic Press, (1976), pp.

3. _____, Parallel algorithms for adaptive quadrature II - Metalgorithm correctness, Acta Information, 4 (1975), pp.

4. _____, Parallel algorithms for adaptive quadrature III - Program correctness, ACM Trans. Math. Software, 2 (1976), pp.

5. Ryder, B.G. , The PFORT verifier, Software Practice and Experience, 4 (1974), pp. 359-377.