Improved Interval Bounds for Ranges of Functions

L. B. Rall

Mathematics Research Center
University of Wisconsin-Madison
610 Walnut Street

Madison, Wisconsin 53705/USA

1. Ranges of functions. The range of a real function $f: D \subset \mathbb{R} \to \mathbb{R}$ on a set $X \subset D$ is

$$R(f;X) = \{f(x) \mid x \in X\}. \tag{1.1}$$

In case $X = [a,b]$ is a closed, bounded interval and f is continuous, then $R(f;X)$ will also be an interval of the same kind. Closed, bounded intervals will be referred to simply as intervals, and the set of such intervals will be denoted by $\mathbb{IR}$.

A fundamental problem of interval analysis is the calculation of $R(f;X)$ or at least a good approximation to it. If f is defined in terms of arithmetic operations and functions with known interval extensions, then straightforward use of interval computation gives an interval extension F of f such that

$$R(f;X) \subset F(X) \tag{1.2}$$

for $X \subset D$. This calculation has the advantage of being completely automatic, and does not require knowledge of special properties of f. Unfortunately, $F(X)$ can be such a gross overestimation of $R(f;X)$ in certain cases that it is useless for practical purposes. Furthermore, the quality of $F(X)$ as an approximation to $R(f;X)$ is generally unknown.

A number of methods have been developed for obtaining better approximations to $R(f;X)$, starting with the work of Moore [1]. The recent book by Ratschek and Rokne [5] describes a number of these techniques, and gives a substantial bibliography. Most of the approaches to this problem are based on transformation of F, usually into centered or mean-value forms [1], [5]. The method given in this paper applied to continuously differentiable functions f, and makes use of information about the monotonicity of f obtained by the process of automatic differentiation [2].

<u>2. Monotone functions.</u> If the function f is nondecreasing on X, then R(f;X) is simply

$$R(f;X) = [f(a), f(b)]. \tag{2.1}$$

Similarly, if f is nonincreasing on X, then

$$R(f;X) = [f(b), f(a)]. \tag{2.2}$$

Thus, the range of monotone functions can be determined by calculating only two function values. In actual practice, of course, downward rounding of the lower endpoint and upward rounding of the upper endpoint gives an interval inclusion of R(f;X) which is slightly wider than the exact range. For the time being, it will be assumed that function values are computed exactly.

A sufficient condition for (2.1) to hold for differentiable f is that

$$f'(x) \geq 0, \quad a \leq x \leq b, \tag{2.3}$$

and similarly (2.2) holds if $f'(x) \leq 0$ on X. Furthermore, suppose that f is continuously differentiable, and F' denotes an interval extension of f' obtained by interval computation. If $F'(X) \geq 0$ ($F'(X) \leq 0$), it follows that f is nondecreasing (nonincreasing) on X, and R(f;X) can be calculated directly by (2.1) or (2.2), respectively.

The additional information about the derivative of f needed above can also be obtained automatically. The values of F(X) and F'(X) can be computed by using interval differentiation arithmetic, as described below. All that is required is a formula or subroutine for f; no symbolic differentiation is necessary. If necessary, a bisection procedure can be applied to the interval X to find subintervals on which f can be guaranteed to be monotone. The resulting algorithm provides either the exact value of R(f;X), or else an inclusion of R(f;X) which is better in general than F(X).

<u>3. Real differentiation arithmetic.</u> It is convenient to define interval differentiation arithmetic as an extension of real differentiation arithmetic. This arithmetic can be used to calculate the values of functions and their derivatives automatically, without symbolics or numerical approximations [4]. Like interval arithmetic, real differentiation arithmetic is an ordered-pair arithmetic, with elements $U = (u,u')$, $V = (v,v')$, ... ε $\mathbf{R}^2$. The rules for this arithmetic are:

$$U + V = (u,u') + (v,v') = (u + v, u' + v'), \tag{3.1}$$

$$U - V = (u,u') - (v,v') = (u - v, u' - v'), \tag{3.2}$$

$$U \cdot V = (u,u') \cdot (v,v') = (u \cdot v,\ u \cdot v' + v \cdot u'), \tag{3.3}$$

$$U/V = (u,u')/(v,v') = (u/v,\ (u' - (u/v) \cdot v')/v), \quad v \neq 0. \tag{3.4}$$

The arithmetic defined in this way forms a division ring with identity, and will be denoted by **D**. If the first element of each operand pair is interpreted as a function value, and the second as a derivative value, then the first element of the result corresponds to the evaluation of the operation, and the second to the evaluation of its derivative, according to the well-known rules of calculus. If real numbers c are identified with the pairs (c,0), then it follows from the chain rule of calculus that

$$f((x,1)) = (f(x),\ f'(x)), \tag{3.5}$$

that is, the rules of differentiation arithmetic will automatically give both the value and the value of the derivative of a rational function f. More generally, the chain rule gives

$$f((u,u')) = (f(u),\ u' \cdot f'(u)), \tag{3.6}$$

which allows the definition of standard functions in D, for example,

$$e^U = e^{(u,u')} = (e^u,\ u' \cdot e^u), \tag{3.7}$$

$$\ln U = \ln(u,u') = (\ln u,\ u'/u), \tag{3.8}$$

and so on. The combination of arithmetic operations and standard functions will be called a computational system for differentiation arithmetic. It is simple to program such a computational system, particulary in a language such as Pascal-SC, which permits definition of operators and functions for various data types [3].

<u>4. Interval differentiation arithmetic.</u> Interval differentiation arithmetic is defined by the same rules as real differentiation arithmetic, starting with pairs of intervals instead of real numbers, and using interval arithmetic instead of real arithmetic inside the parentheses on the right sides of (3.1)-(3.3). With interval extensions of standard functions, the definitions (3.7), (3.8) and so on are used to construct a computational system for interval differentiation arithmetic. Once again, such a system is easy to program in Pascal-SC, which supports interval arithmetic as well as operator and function definitions for various data types [3].

The analog to (3.5) in interval differentiation arithmetic is

$$F((X,[1,1])) = (F(X),\ F'(X)). \tag{4.1}$$

Thus, by a direct evaluation process in this arithmetic, interval inclusions $F(X)$ of $R(f;X)$ and $F'(X)$ of $R(f';X)$ can both be obtained automatically. Here, even if $F'(X)$ is a crude approximation to the range of f' on X, the conditions $F'(X) > 0$ or $F'(X) < 0$ are sufficient to guarantee the monotonicity of f, and if f is monotone, then its range can be calculated exactly by (2.1) or (2.2). This observation is the basis of the algorithm described in the next section.

5. An algorithm for range calculation. Of course, if the calculation of $F'(X)$ shows that f is monotone on the entire interval X, then $R(f;X)$ can be calculated at once. Otherwise, X will be partitioned into subinterval, and either $R(f;X)$ or an approximation to it will be constructed. Let a given list of n subintervals of X be denoted by $L_n = \{X_1, X_2, \ldots, X_n\}$, and suppose that $R \subset R(f;X)$ is known. On each subinterval X_i, either $F(X_i) \subset R$, in which case $R(f;X_i)$ makes no additional contribution to $R(f;X)$, or f is monotone, in which case its range can be computed directly and R updated, or else 0 is an interior point of $F'(X_i)$, in which case X_i may contain a critical point of f. In the latter case, X_i can be bisected and the resulting subintervals put on a new list for further examination. In order for the algorithm to terminate in a finite number of steps, a lower bound δ is put on the widths of the subintervals to be considered, and an upper bound N is placed on the number of subintervals to be saved for further examination. For convenience, if Y,Z are intervals, then $Y ++ Z$ will denote the interval hull of Y and Z, that is, the smallest interval which contains both Y and Z.

The algorithm consists of the following steps:

1°. (Initialization) Take $X_1 := X$, $L_1 := \{X_1\}$, $R := [f(x), f(x)]$, where x is some point in X.

2°. (Iteration) For $i = 1,\ldots,n$, compute $(F(X_i), F'(X_i))$

(a) If $F(X_i) \subset R$, then discard X_i.

(b) If $F'(X_i) > 0$ or $F'(X_I) < 0$, then compute $R := R ++ R(f,X_i)$ and discard X_i.

(c) Otherwise, retain X_i.

3°. (Termination or continuation) Denote the list of retained intervals by L_r.

(a) If L_r is empty, then the algorithm terminates with the exact value

$$R = R(f;X) \tag{5.1}$$

of the range of f on X.

(b) If $r > N$ or $w(X_1) < \varepsilon$, then the algorithm terminates with the overestimate

$$R := R \mathbin{++} F(X_1) \mathbin{++} \dots \mathbin{++} F(X_r) \supset R(f;X) \tag{5.2}$$

of the range of f on X.

(c) Otherwise, each subinterval in L_r is bisected to form a new list L_n with $n = 2r$, and the algorithm returns to step 2°.

6. Remarks. The algorithm given in the previous section will terminate in a finite number of steps with either the exact value of $R(f;X)$ or an overestimate which is never worse than

$$R = F(X_1) \mathbin{++} \dots \mathbin{++} F(X_n) \supset R(f;X). \tag{6.1}$$

In general, (6.1) is a better approximation to $R(f;X)$ than $F(X)$ because of the convergence of united extensions to the range of a continuous function [1].

As a byproduct of the calculation when an overestimate is produced, the intervals $X_1,\dots,X_r$ which are retained at the final step may contain critical points of f, that is, points at which $f'(x) = 0$. This information may be useful in optimization problems. Furthermore, if the list of retained intervals is nonempty, then the value $R \supset R(f;X)$ returned by the algorithm is definitely known to be an overestimate, while if the list of retained algorithms is empty, then this value is exact (modulo outward rounding). Thus, the algorithm itself indicates the type of result (exact or an overestimate) it obtains. The knowledge that R is an overestimate and the list of retained intervals can be used to refine the calculation of $R(f;X)$ further, if desired. Some idea of the quality of the overestimate can be obtained by comparing the value of R before calculating (5.2) with the final result.

7. Numerical results. Numerical results were computed for the following functions, using the Pascal-SC program given in the following section.

$$f_1(x) = x - x, \tag{7.1}$$

$$f_2(x) = x \cdot x, \tag{7.2}$$

$$f_3(x) = \frac{(x - 1)\cdot(x + 3)}{(x + 2)}, \tag{7.3}$$

$$f_4(x) = x/x. \tag{7.4}$$

a. For $X = [a,b]$, the naive interval extension $F_1(X) = X - X$ of f_1 gives $F_1([a,b]) = [a - b, b - a]$, while the algorithm gives $R = [0,0] = R(f_1;X)$ for arbitrary X.

b. For symmetric intervals $X = [-s,s]$, the algorithm gives the exact value $R = [0,s^2] = R(f_2; [-s,s])$, while $F_2([-s,s]) = [-s,s]\cdot[-s,s] = [-s^2,s^2]$. In case $X = [-r,s]$ is nonsymmetric interval containing 0, the result of the algorithm can be of the form $R = [-\varepsilon, \max\{r^2,s^2\}$, where $\varepsilon > 0$ is small, with a message that a small interval containing 0 can contain a critical point of f_2. For example, for $X = [-7,8]$, one has

$$F_2(X) = X\cdot X = [-56, 64] \tag{7.5}$$

while the algorithm gives

$$R = [-3.1\times10^{-18}, 64] \tag{7.6}$$

with a notation that there may be a critical point of f_2 in the retained interval $[-1.63\times10^{-9}, 1.87\times10^{-9}]$. In all other cases, the algorithm gives the exact result. Even if X is nonsymmetric about 0, the algorithm will give the correct result if 0 is a bisection point.

c. The function f_3 is actually monotone increasing, but has a pole at $x = -2$. The algorithm will sense the monotonicity of f_3 and give correct results if X is subdivided a sufficient number of times. The results are much better than the naive interval extension $F_3(X) = (X - 1)\cdot(X + 3)/(X + 2)$ when one of the endpoints of X is close to -2. For example, for $X = [-1.9, 98]$,

$$F_3(X) = [-2929, 97970], \tag{7.7}$$

while the algorithm gives

$$R = [-31.9, 97.97]. \tag{7.8}$$

For $X = [-1.999999, 98]$,

$$F_3(X) = [-3.03\times10^8, 9.797\times10^9], \tag{7.9}$$

while the algorithm gives

$$R = [-3000002, 97.97]. \tag{7.10}$$

Finally, for $X = [-1.99999999999, 98]$, which has a lower endpoint as close to -2 as possible in 12-digit decimal arithmetic, one gets

$$F_3(X) = [-3.03\times10^{13}, 9.797\times10^{14}], \tag{7.11}$$

while the algorithm gives

$$R = [-300000000002,\ 97.97] \tag{7.12}$$

d. The algorithm does not give good results for $f_4(x) = x/x$, because it determines that every subinterval of X possibly contains a critical point of f_4 (which in fact is true, since $f_4(x) \equiv 1$ is constant, and $f_4'(x) \equiv 0$). Thus, the algorithm computes $R = [1,1]$ initially, and the final value is determined only by the united extension (6.1). Of course, the result is generally better than the naive interval extension $F_4(X) = X/X$ evaluated on the entire interval X, but is still usually a gross overestimate. For example, for $X = [0.002, 2]$.

$$F_4(X) = [0.001,\ 1000] \tag{7.13}$$

while the algorithm gives

$$R = [0.203,\ 4.903], \tag{7.14}$$

which is still not a very good approximation to [1,1], even though it is much better than (7.13). Of course, the user is warned that the result may not be good by the fact that all subintervals are retained. Other methods usually give no warning when gross overestimates are produced. One way to improve the algorithm in this case, since interval extensions of derivatives are available, would be to use mean-value forms

$$F(X_i) = m(X_i) + F'(X_i)\cdot(X - m(X_i)) \tag{7.15}$$

to obtain interval extensions F of f on subintervals X_i, instead of obtaining them by straightforward evaluation.

8. A Pascal-SC program. The program written below was designed to be general, so that the user needs to supply only subroutines for evaluation of the function f in ordinary interval arithmetic (IFEVAL) and in interval differentiation arithmetic (IDFEVAL). The source code for these subroutines should be located in the files FEVAL.FUN. Examples of these subroutines for the functions discussed in §7 are given in §10.

The operators for interval differentiation arithmetic given in §9 include only the basic arithmetic operators for type IDERIV. For a complete computational system, operators for mixed arithmetic between types INTEGER, REAL, and IDERIV should be included, as well as standard functions [3].

The number of subintervals allowed in a list is set by the constant DIM in the program, which can be changed by the user. The size of the smallest subintervals is similarly controlled by the number LIMIT of bisections allowed. Thus, if LIMIT = L, then

$$\delta = 2^{-L} \cdot w(X), \tag{8.1}$$

where $w(X) = b - a$ is the width of the original interval $X = [a,b]$. The source code for the Pascal-SC program follows:

```
PROGRAM IRANGE(INPUT,OUTPUT);
 CONST DIM = 256;      (* Maximum number of subintervals *)
       LIMIT = 32;     (* Maximum number of bisections *)
 TYPE INTERVAL = RECORD INF,SUP : REAL END ;
      IDERIV = RECORD X,PRIME: INTERVAL END;
      DIMTYPE = 1..DIM;
      STACKTYPE = RECORD INT:INTERVAL;FUN:IDERIV END;
 VAR X,RF,BEST,WORST: INTERVAL;
     F: IDERIV;
     I,NA,NB,LIM: INTEGER;
     A,B: ARRAY[DIMTYPE]OF STACKTYPE;  (* A is the list of intervals to
                                          be examined, B is the list of
                                          retained intervals *)
     MX: REAL;
$INCLUDE INTERVAL.PAK;       (* Makes interval arithmetic available *)
$INCLUDE IDERV.PAK;          (* Interval differentiation arithmetic *)
PROCEDURE IOUT(X: INTERVAL); (* Prints endpoints in standard format *)
 BEGIN
  WRITE('[',X.INF,',',X.SUP,']');
 END;
 $INCLUDE FEVAL.FUN;     (* Evaluation of the function in interval and
                            and interval differentiation arithmetic *)
FUNCTION RMF(L,G: REAL): INTERVAL;
 (* Bounds the range of a monotone function which assumes its least
    value at L and its greatest value at G. *)
 VAR D,U: INTERVAL;
 BEGIN
  D:=INTPT(L);U:=INTPT(G);
  D:=IFEVAL(D);U:=IFEVAL(U);
  D.SUP:=U.SUP;
  RMF:=D
 END;
```

```
FUNCTION MID(X: INTERVAL): REAL;  (* Calculates midpoint of an interval *)
 VAR A,B: ARRAY[1..2]OF REAL;
 BEGIN
  A[1]:=X.INF;B[1]:=0.5;
  A[2]:=X.SUP;B[2]:=0.5;
  MID:=SCALP(A,B,0)
 END;

BEGIN  (* Program IRANGE *)
 WRITELN('Enter initial interval X:');
 IREAD(INPUT,X);
 WRITE('    X = ');IOUT(X);WRITELN;
 F:=IDFEVAL(X);
 WORST:=F.X;
 IF (F.PRIME.INF >= 0) THEN RF:=RMF(X.INF,X.SUP)
 ELSE IF (F.PRIME.SUP <= 0) THEN RF:=RMF(X.SUP,X.INF)
 ELSE
 BEGIN  (* F is not monotone *)
  NA:=1;LIM:=0;
  A[1].INT:=X;A[1].FUN:=F;
  MX:=MID(X);
  X:=INTPT(MX);
  BEST:=IFEVAL(X);
  WHILE ((NA > 0) AND (NA <= DIM DIV 2) AND (LIM < LIMIT)) DO
  BEGIN (* WHILE *)
   LIM:=LIM+1;NB:=0;
   FOR I:=1 TO NA DO
    BEGIN (* STACK B *)
     MX:=MID(A[I].INT);
     NB:=NB+1;
     B[NB].INT.INF:=A[I].INT.INF;
     B[NB].INT.SUP:=MX;
     B[NB].FUN:=IDFEVAL(B[NB].INT);
     NB:=NB+1;
     B[NB].INT.INF:=MX;
     B[NB].INT.SUP:=A[I].INT.SUP;
     B[NB].FUN:=IDFEVAL(B[NB].INT);
    END;  (* STACK B *)
    NA:=0;
    FOR I:=1 TO NB DO
     BEGIN (* UNSTACK B *)
```

```
      IF NOT (B[I].FUN.X <= BEST)
      THEN IF (B[I].FUN.PRIME.INF >= 0)
      THEN BEST:=BEST+*RMF(B[I].INT.INF,B[I].INT.SUP)
      ELSE IF (B[I].FUN.PRIME.SUP <= 0)
      THEN BEST:=BEST+*RMF(B[I].INT.SUP,B[I].INT.INF)
      ELSE
       BEGIN (* RESTACK A *)
        NA:=NA+1;A[NA]:=B[I]
       END;  (* RESTACK A *)
     END;  (* UNSTACK B *)
     RF:=BEST;
     FOR I:=1 TO NA DO RF:=RF+*A[I].FUN.X;
  END;  (* WHILE *)
  IF NA > 0 THEN
  BEGIN (* NA > 0 *)
   RF:=BEST;
   WRITELN('Function may have critical points in:');
   FOR I:=1 TO NA DO
   BEGIN
    WRITE('A[',I:2,'] = ');IOUT(A[I].INT);WRITELN;
    RF:=RF+*A[I].FUN.X
   END;
  END;  (* NA > 0 *)
 END;  (* F is not monotone *)
 WRITELN('Naive interval arithmetic gives:');
 WRITE(' F(X) = ');IOUT(WORST);WRITELN;
 WRITELN('The algorithm gives:');
 WRITE(' F(X) = ');IOUT(RF);WRITELN
END.  (* Program IRANGE *)
```

9. The operators for interval differentiation arithmetic. The six basic unary and binary arithmetic operators for type IDERIV are located in the file IDERIV.PAK, which also includes the call to the interval library for the function ISCALP to compute the interval scalar product.

```
TYPE IVECTOR = ARRAY[1..2]OF INTERVAL;
FUNCTION ISCALP ( VAR A,B: IVECTOR; DIM: INTEGER): INTERVAL;
 EXTERNAL 88;      (* Interval scalar product *)
OPERATOR + (U: IDERIV) RES: IDERIV;
 BEGIN
  RES:=U
 END;
```

```
OPERATOR - (U: IDERIV) RES: IDERIV;
 BEGIN
  U.X:=-U.X;
  U.PRIME:=-U.PRIME;
  RES:=U
 END;
OPERATOR + (U,V: IDERIV) RES: IDERIV;
 BEGIN
  U.X:=U.X+V.X;
  U.PRIME:=U.PRIME+V.PRIME;
  RES:=U
 END;
OPERATOR - (U,V: IDERIV) RES: IDERIV;
 BEGIN
  U.X:=U.X-V.X;
  U.PRIME:=U.PRIME-V.PRIME;
  RES:=U
 END;
OPERATOR * (U,V: IDERIV) RES: IDERIV;
 VAR A,B: IVECTOR;
 BEGIN
  A[1]:=U.X;B[1]:=V.PRIME;
  A[2]:=V.X;B[2]:=V.PRIME;
  U.PRIME:=ISCALP(A,B,2);
  U.X:=U.X*V.X;
  RES:=U
 END;

OPERATOR / (U,V: IDERIV) RES: IDERIV;
 VAR A,B: IVECTOR;
     C: IDERIV;
 BEGIN
  C.X:=U.X/V.X;
  A[1]:=INTPT(1);B[1]:=U.PRIME;
  A[2]:=-C.X;B[2]:=V.PRIME;
  C.PRIME:=ISCALP(A,B,2)/V.X;
  RES:=C
END;
```

10. Example function subroutines. (Contents of the file FEVAL.FUN.)

(a) $f_1(x) = x - x$.

```
FUNCTION IFEVAL(X: INTERVAL): INTERVAL;
 BEGIN
  IFEVAL := X - X
 END;
FUNCTION IDFEVAL(X: IDERIV): IDERIV;
 BEGIN
  IDFEVAL := X - X
 END;
```

(b) $f_2(x) = x \cdot x$.

```
FUNCTION IFEVAL(X: INTERVAL): INTERVAL;
 BEGIN
  IFEVAL := X*X
 END;
FUNCTION IDFEVAL(X: IDERIV): IDERIV;
  BEGIN
   IDFEVAL := X*X
  END
```

(c) $f_3(x) = (x - 1)\cdot(x + 3)/(x + 2)$.

```
FUNCTION IFEVAL(X: INTERVAL): INTERVAL;
 BEGIN
  IFEVAL := (X - 1)*(X + 3)/(X + 2)
 END;
FUNCTION IDFEVAL(X: IDERIV): IDERIV;
 VAR ONE,TWO,THREE: IDERIV;
 BEGIN
  ONE.X := INTPT(1);ONE.PRIME := INTPT(0);
  TWO.X := INTPT(2);TWO.PRIME := INTPT(0);
  THREE.X := INTPT(3);THREE.PRIM := INTPT(0);
  IDFEVAL := (X - ONE)*(X + THREE)/(X + TWO)
 END;
```

(d) $f_4(x) = x/x$.

```
FUNCTION IFEVAL(X: INTERVAL): INTERVAL;
 BEGIN
  IFEVAL := X/X
 END;
FUNCTION IDFEVAL(X: IDERIV): IDERIV;
 BEGIN
```

```
IDFEVAL := X/X
END;
```

11. Acknowledgements. This work was sponsored by the United States Army under Contract No. DAAG29-80-C-0041. The author is grateful to Prof. Karl Nickel for helpful comments and suggestions.

References

1. R. E. Moore. Interval Analysis, Prentice-Hall, Englewood Cliffs, N. J., 1966.

2. L. B. Rall. Automatic Differentiation: Techniques and Applications. Lecture Notes in Computer Science No. 120, Springer, New York, 1981.

3. L. B. Rall. Differentiation and generation of Taylor coefficients in Pascal-SC, pp. 291-309 in A New Approach to Scientific Computation, ed. by U. W. Kulisch and W. L. Miranker, Academic Press, New York, 1983.

4. L. B. Rall. The Arithmetic of Differentiation. MRC Technical Summary Report No. 2688, Mathematics Research Center, University of Wisconsin-Madison, May, 1984. To appear in Mathematics Magazine, February, 1986.

5. H. Ratschek and J. Rokne. Computer Methods for the Ranges of Functions. Halsted Press, Wiley, New York, 1984.