Chapter 1
Automatic Differentiation of Conditional Branches in an Operator Overloading Context*

R. Baker Kearfott

February 23, 1996

Abstract

In the past, it has been problematical to include `IF-THEN-ELSE` branches in automatic differentiation processes driven by operator overloading and code list generation, when the branch condition contains variables. However, this problem can be circumvented with a special "branch function" χ. Definition of this function, formulas for its use, and implications of its use will be discussed.

A second issue is: what can be done when derivatives are discontinuous? In fact, simple and meaningful Newton iterations can be set up when even the function itself is discontinuous. Simplified figures and examples are given, as well as references to in-depth explanations. An example of the convergence behavior is given with an interval Newton method to find critical points for the problem "$\min |x|$."

1 Introduction and Context

Consider:

$$\phi(x) = \max(2 - x^2, x^2). \tag{1}$$

Operator overloading may be used to generate a *code list*, i.e. an internal representation of the function [9]. This internal representation can then be used to obtain function values and derivatives. For example, Example (1) would be programmed in the Fortran 90 system described in [3] and [6, §2.2.2] as:

```
   PROGRAM EXAMPLE
     USE OVERLOAD
     IMPLICIT NONE
       TYPE(CDLVAR), DIMENSION(1):: X
       TYPE(CDLLHS), DIMENSION(1):: PHI

       OUTPUT_FILE_NAME='EX6_P1.CDL'
       CALL INITIALIZE_CODELIST(X)

       PHI(1) = MAX(2-X(1)**2,X(1)**2)

       CALL FINISH_CODELIST
 END PROGRAM EXAMPLE
```

In this program, the independent variable `X` and dependent variable `PHI` are declared to be a special data type. The arithmetic operations and function evaluations take on a special

*This work was supported in part by National Science Foundation grant CCR-9203730.

meaning: Instead of doing arithmetic operations, the computer writes a numerical code that identifies the operation and numbers that identify the operands to a line of a file (in this case, the file `EX6_P1.CDL`). The result of executing the program is the code list, or ordered list of operations to evaluate ϕ. With the NAG Fortran compiler, version 2.1, and the system of [3], the program produces a code list that signifies the following:

	x_1	$=$	x
1.	x_2	$\leftarrow$	x_1^2
2.	x_3	$\leftarrow$	$-x_2 + 2$
3.	x_4	$\leftarrow$	x_1^2
4.	x_5	$\leftarrow$	$\max x_3, x_4$
	ϕ	$=$	x_5

This list corresponds to the order in which the compiler sets up the operations, including any optimization.

Alternate automatic differentiation procedures are:

1. overloading the operations so that both the function and derivative are computed *numerically* as the program is executed;
2. designing special-purpose compilers.

Production of code lists makes it somewhat easier, relative to direct computation as the function executes, to implement a large number of different computations, such as derivatives of various orders with different data types, and allows post-processing of the code list for additional optimization. Production of code lists through operator overloading allows use of existing compiler technology, including optimization, thus avoiding some duplication of effort necessary if a special purpose-compiler were designed.

Although some programs may produce sizeable code lists, the operator overloading/code list technique can handle looping instructions and subroutine calls without complications. However, conditional transfer of control, such as in `IF-THEN-ELSE` constructs, poses more of a problem, since the operations that are executed then depend on the *numerical* value of the independent variables. Consider:

$$f(x) = \begin{cases} -2x & \text{if} \quad x^2 - 1 < 0 \\ 2x & \text{if} \quad \text{otherwise.} \end{cases} \tag{2}$$

We would like the Fortran 90 code corresponding to the line "`PHI(1) = ...`" above to be:

```
 IF (X**2-1D0 < 0) THEN
    F = -2D0*X
ELSE
    F =  2D0*X
END IF
```

However, there is no way for both branches to be executed (and thus be represented in a code list) by a simple execution of the program. A simple solution is reviewed in §2.

When max, min, and $|\circ|$ appear in objective functions, such as in Example (1), derivatives can be discontinuous. In functions such as Example (2), the function itself is discontinuous at a branch point. There is a question of what values should be assigned to derivatives at a branch point. There is a simple and logical solution to this problem if interval arithmetic is used. In fact, a simple Newton iteration can even be applied to f of Example (2). This is reviewed in §3.

The remainder of this paper presents various examples, computations, simplified figures, and perspective. A complete set of formulas, as well as experiments with nonlinear equations and optimization problems in various dimensions, appears in [4]. A convergence analysis of interval Newton methods with discontinuous derivatives appears in [5]. A unified explanation, including a discussion of automatic differentiation and the Fortran 90 system we use, will appear in [6].

2 The Branching Function

Define

$$\chi(x_s, x_q, x_r) = \begin{cases} x_q & \text{if } x_s < 0, \\ x_r & \text{otherwise.} \end{cases}$$

The function f of Example (2) would then be programmed as

```
F = CHI(X**2-1,-2D0*X,2D0*X)
```

The program containing this statement would then produce a code list that contains *both* branches. In fact, the system of [3], compiled with the NAG Fortran 90 compiler version 2.1, produces a code list that signifies:

	x_1	$=$	x
1.	x_2	$\leftarrow$	x_1^2
2.	x_3	$\leftarrow$	$x_2 - 1$
3.	x_4	$\leftarrow$	$2x_1$
4.	x_5	$\leftarrow$	$-x_4$
5.	x_6	$\leftarrow$	$2x_1$
6.	x_7	$\leftarrow$	$\chi(x_3, x_5, x_6)$
	f	$=$	x_7

The function χ can be differentiated and evaluated using the formulas of [4] or [6, Chapter 6]. Hence, functions f whose evaluation contains branches can be differentiated in an operator overloading context; see [3] or [6, §1.4], for example.

3 On Non-Smooth and Discontinuous Derivatives

The functions $|\circ|$, max, and min are important in optimization. The derivatives of these functions can be expressed in terms of χ. For example:

$$\frac{\partial \max\{u, v\}}{\partial x} = \chi\left(v - u, \frac{\partial u}{\partial x}, \frac{\partial v}{\partial x}\right). \tag{3}$$

(Complete sets of formulas can be found in [4] or [6].)

3.1 Non-Smooth Functions

Equation (3) and analogous formulas for $|\circ|$ and min can be used for correct evaluation of derivatives when $v - u$ is non-zero (or when $v - u$ does not contain zero, in the case of interval data). However, there has been a standing question of what to do at the branch point itself (i.e. where $v - u = 0$). This question can be resolved practically and simply with interval arithmetic. Denoting intervals by boldface, interval evaluation of χ is:

$$\chi(\boldsymbol{x}_s, \boldsymbol{x}_q, \boldsymbol{x}_r) = \begin{cases} \boldsymbol{x}_q & \text{if } \boldsymbol{x}_s < 0; \\ \boldsymbol{x}_r & \text{if } \boldsymbol{x}_s > 0; \\ \boldsymbol{x}_q \underline{\cup} \boldsymbol{x}_r & \text{otherwise.} \end{cases} \tag{4}$$

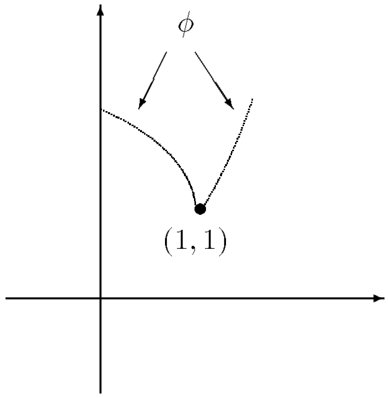

FIG. 1. *Graph of* $\phi(x) = \max(2 - x^2, x^2)$

This interval value contains the set of all values $\chi(x_s, x_q, x_r)$ as x_s ranges in $\boldsymbol{x}_s$, x_q ranges in $\boldsymbol{x}_q$, and x_r ranges in $\boldsymbol{x}_r$.

Even when most of the computations are done in real arithmetic, the interval formula can still provide derivatives at branch points. For example, if ϕ is as in Example (1), then

$$\frac{\partial \phi}{\partial x} = \chi\Big(x^2 - (2 - x^2), -2x, 2x\Big),$$

and $\partial\phi/\partial x|_{x=1}$ may be set to the interval value

$$\chi\Big([1,1], -2[1,1], 2[1,1]\Big) = [-2, 2].$$

A very useful aspect of these computations is that critical points of optimization problems with objective functions ϕ may still be sought by seeking $\nabla\phi = 0$. However, in this case, the critical point may happen to be a point where the derivative is undefined. In the above, example, $0 \in [-2, 2]$, so the "derivative" contains zero, thus indicating that $x = 1$ represents a critical point. This is illustrated in the graph of ϕ in Figure 1. (Note that, in this case, the interval extension corresponds to the notion of subgradient.)

Formula (4) also provides a solution to the "if problem" mentioned by Beck and Fischer [2]. In particular, if the standard formula

$$\frac{\partial \chi(w, u, v)}{\partial x} = \chi\left(w, \frac{\partial u}{\partial x}, \frac{\partial v}{\partial x}\right)$$

is used to differentiate a branch, then evaluation with Formula (4) results in an answer that always contains the correct derivative, even for functions that are not derivative-consistent in the sense in [2]. In fact, this technique can even be used in floating-point computations, with intervals entering only where necessary.

Observe that *both* branches are stored symbolically in this scheme. This requires more storage for the code list (also known as the "tape" or "directed acyclic graph"), but provides additional flexibility. In many instances, the additional storage is not prohibitive. Also, both values of the branches are used only in special instances. It may be efficient to actually evaluate both branches, setting undefined expressions to the real line or the empty set, as appropriate. This is definitely true in certain interval computations.

Alternatives to storing both branches are "retaping" (a technique used by Griewank et al in ADOLC), or using a precompiler, rather than operator overloading. There are advantages and disadvantages to both these approaches.

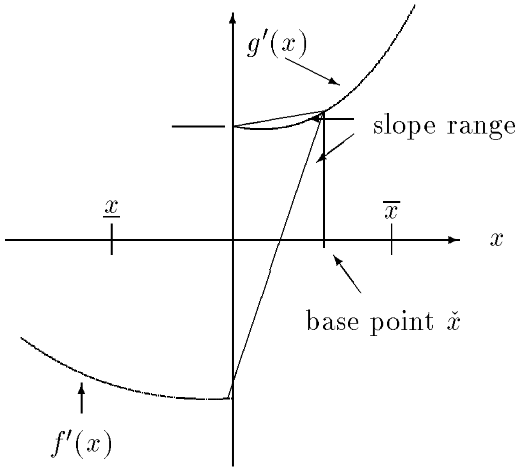

FIG. 2. *The slope of* $\chi((g-f)(x), f'(x), g'(x))$

3.2 Derivatives of Discontinuous Functions

If $\phi(x) = \max(2 - x^2, x^2)$ is as in Example (1), then $\nabla\phi$ is discontinuous at $x = 1$, but, as seen above, a meaningful interpretation can be given to $\nabla\phi$ at the discontinuity. But what about *second derivatives* of ϕ, needed, for example, if we were to locate critical points of ϕ with Newton's method? In fact, an interval Newton method can be used precisely in this context, if *interval slopes*, rather than actual derivatives, are used. An interval slope $\mathbf{S}(f, \boldsymbol{x}, \check{x})$ of a function f from a base point $\check{x}$ to an interval $\boldsymbol{x}$ is a set that contains the set of all slopes of lines from the point $(\check{x}, f(\check{x}))$ to points $(x, f(x))$, for $x \in \boldsymbol{x}$. Slopes were introduced and analyzed in [1, 7], among others. The book [8] contains a good theoretical presentation. The book [6] contains an introduction, discussion, examples, and methods of computation for univariate and multivariate slopes, as well as additional references.

For example, the derivative of $\max\{f(x), g(x)\}$ is $\chi\Big((g-f)(x), f'(x), g'(x)\Big)$; the slopes of this derivative are illustrated in Figure 2.

A formula for computing interval slopes of such a discontinuous function, when $0 \in \boldsymbol{x}_s \underline{\cup} \check{\boldsymbol{x}}_s$ (as in Figure 2), is:

$$\mathbf{S}(\chi(x_s(x), x_q(x), x_r(x)), \boldsymbol{x}, \check{x}) =$$

$$(5)\qquad \begin{cases} \mathbf{S}(x_r(x), \boldsymbol{x}, \check{x}) \underline{\cup} \left\{ \frac{x_r(x) - x_q(x)}{\left[x_s(\check{x}), x_s(\check{x}) - \underline{x}_s(x)\right]} \mathbf{S}(x_s(x), \boldsymbol{x}, \check{x}) \right\} & \text{if } \boldsymbol{x}_s(\check{x}) > 0 \\[2ex] \mathbf{S}(x_q(x), \boldsymbol{x}, \check{x}) \underline{\cup} \left\{ \frac{x_r(x) - x_q(x)}{\left[-x_s(\check{x}), \overline{x}_s(x) - x_s(\check{x})\right]} \mathbf{S}(x_s(x), \boldsymbol{x}, \check{x}) \right\} & \text{if } \boldsymbol{x}_s(\check{x}) < 0 \\[2ex] \left\{ \left[\frac{1}{x_s(\check{x}) - \underline{x}_s(x)}, \infty \right) \cup \left[\frac{1}{\overline{x}_s(x) - x_s(\check{x})}, \infty \right) \right\} (\boldsymbol{x}_r(\boldsymbol{x}) - \boldsymbol{x}_q(\boldsymbol{x})) \cdot & \\ \qquad \mathbf{S}(x_s(x), \boldsymbol{x}, \check{x}) & \\ \qquad\qquad \underline{\cup} \mathbf{S}(x_q(x), \boldsymbol{x}, \check{x}) \underline{\cup} \mathbf{S}(x_r(x), \boldsymbol{x}, \check{x}) & \text{if } 0 \in \boldsymbol{x}_s(\check{x}). \end{cases}$$

(Here, "$\boldsymbol{x} \underline{\cup} \boldsymbol{y}$" represents the convex hull of the intervals $\boldsymbol{x}$ and $\boldsymbol{y}$.) This formula appeared, slightly generalized and stated for partial derivatives, in [4, 5], and can be found, with an explanation and complete context, in [6]. The formula in [5] contains different denominators, rather than $[\underline{x}_s(\check{\boldsymbol{x}}), \overline{x}_s(\check{\boldsymbol{x}}) - \underline{x}_s(\boldsymbol{x})]$, etc.; that leads to somewhat narrower widths, but is incorrect for various cases.

TABLE 1
Interval Newton iterates for $f(x) = \chi(x^2 - 1, -2x, 2x)$

#	interval	rel. width
1	$[1.000, 5.000]$	0.80
2	$[1.000.2.700]$	0.63
3	$[1.000, 1.668]$	0.40
4	$[1.000, 1.230]$	0.19
5	$[1.000, 1.068]$	0.64E-01
6	$[1.000, 1.019]$	0.18E-01
7	$[1.000, 1.005]$	0.46E-02
8	$[1.000, 1.002]$	0.12E-02
9	$[1.000, 1.001]$	0.29E-04

We apply Formula (5) and other formulas for the code list, as explained in [6], to $\nabla\phi = \chi(x^2 - 1, -2x, 2x)$. Starting with base point $\check{x} = 3$ and $\boldsymbol{x} = [1, 5]$, an interval Newton method gives:

$$\begin{aligned} \tilde{\boldsymbol{x}} &= \check{x} - \frac{\boldsymbol{f}(\check{x})}{\mathbf{S}(\boldsymbol{f}, \boldsymbol{x}, \check{x})} \\ &= 3 - \frac{6}{[2, 20]} \\ &= [0, 2.7], \end{aligned}$$

and $\boldsymbol{x} \leftarrow \boldsymbol{x} \cap \tilde{\boldsymbol{x}} = [1, 2.7]$, a significant reduction in width. A continuation of the interval Newton method gives the results in Table 1. In this table, the actual interval results were rounded outward to four digits; the column labelled "rel. width" gives the width of the interval, divided by the magnitude (maximum absolute value) of the interval, and rounded to two significant figures.

A second example of use of Formula (5) in an interval Newton method reveals underlying convergence properties in the simplest possible setting. Take $\phi(x) = |x|$, so that $\nabla\phi(x) = \chi(x, -1, 1)$, a step function with step of height 2 at $x = 0$. Take $\check{x}$ to be the midpoint: $\check{x} = (\underline{x} + \overline{x})/2$, where $\boldsymbol{x} = [\underline{x}, \overline{x}]$. Assume that $0 \in \boldsymbol{x}$. If $\check{x} > 0$, then Formula (5) gives:

$$\begin{aligned} \mathbf{S}(\chi(x, -1, 1), \boldsymbol{x}, \check{x}) &= 0 \underline{\cup} \left\{ \frac{2}{\left[\frac{\underline{x}+\overline{x}}{2}, \frac{\underline{x}+\overline{x}}{2} - \underline{x}\right]} \right\} \\ &= \left[0, \frac{4}{\underline{x}+\overline{x}}\right] \end{aligned} \tag{6}$$

Similarly, if $\check{x} < 0$, then

$$\mathbf{S}(\chi(x, -1, 1), \boldsymbol{x}, \check{x}) = \left[0, \frac{4}{-(\underline{x} + \overline{x})}\right].$$

For $\check{x} > 0$, the interval Newton method thus becomes:

TABLE 2
Interval Newton iterates for $f(x) = \chi(x, -1, 1)$

#	interval	midpoint	width	$\frac{\text{new width}}{\text{old width}}$
1	$[-1.0000, 8.0000]$	3.50	9.00	—
2	$[-1.0000, 1.7500]$	3.75E-01	2.75	0.31
3	$[-1.0000, 0.1875]$	−4.06E-01	1.19	0.43
4	$[-0.2031, 0.1875]$	−7.81E-03	3.91E-01	0.33
5	[−0.3906E-02, 0.1875E+00]	9.18E-02	1.91E-01	0.49
6	[−0.3906E-02, 0.4590E-01]	2.10E-02	4.98E-02	0.26
7	[−0.3906E-02, 0.1050E-01]	3.30E-03	1.44E-02	0.29
8	[−0.3906E-02, 0.1648E-02]	−1.13E-03	5.55E-03	0.39
9	[−0.5645E-03, 0.1648E-02]	5.42E-04	2.51E-03	0.45
10	[−0.5645E-03, 0.2709E-03]	−1.47E-04	8.35E-04	0.33

$$
\begin{aligned}
\tilde{\boldsymbol{x}} &\leftarrow \frac{\underline{x}+\overline{x}}{2} - \frac{1}{\left[0, \frac{4}{\underline{x}+\overline{x}}\right]} \\
&= \frac{\underline{x}+\overline{x}}{2} - \left[\frac{\underline{x}+\overline{x}}{4}, \infty\right] \\
&= \left[-\infty, \frac{\underline{x}+\overline{x}}{4}\right], \\
\boldsymbol{x} &\leftarrow \tilde{\boldsymbol{x}} \cap \boldsymbol{x} \\
&= \left[\underline{x}, \frac{\underline{x}+\overline{x}}{4}\right]
\end{aligned} \tag{7}
$$

Similarly, for $\check{x} < 0$, the interval Newton method becomes:

$$
\begin{aligned}
\tilde{\boldsymbol{x}} &\leftarrow \left[\frac{\underline{x}+\overline{x}}{4}, \infty\right], \\
\boldsymbol{x} &\leftarrow \tilde{\boldsymbol{x}} \cap \boldsymbol{x} \\
&= \left[\frac{\underline{x}+\overline{x}}{4}, \overline{x}\right]
\end{aligned} \tag{8}
$$

From Equation (7) and Equation (8), we see that *at least half* of the interval $\boldsymbol{x}$ is rejected on each iteration of the interval Newton method. Thus, the convergence is linear, with width-reduction factor less than or equal to 0.5. Furthermore, the process tends to center the critical point in $\boldsymbol{x}$. Starting with $\boldsymbol{x} = [-1, 8]$ (so $\check{x} = 3.5$) results in Table 2.

Acknowledgements I wish to thank Bill Walster for suggesting that it would be possible to proceed with Newton iteration when the function is discontinuous, and for encouraging me to do so.

References

[1] G. Alefeld, *Bounding the slope of polynomial operators and some applications*, Computing, 26 (1980), pp. 227–237.
[2] T. Beck and H. Fischer, *The IF-problem in automatic differentiation*, Journal of Comput. Appl. Math., 50 (1995), pp. 119–131.
[3] R. B. Kearfott, *A Fortran 90 environment for research and prototyping of enclosure algorithms for nonlinear equations and global optimization*, ACM Trans. Math. Software, 21 (1995), pp. 63–78.
[4] ——, *Interval extensions of non-smooth functions for global optimization and nonlinear systems solvers*, 1995. Accepted for publication in *Computing*.

[5] ——, *Treating non-smooth functions as smooth functions in global optimization and nonlinear systems solvers*, in Scientific Computing and Validated Numerics, G. Alefeld and A. Frommer, eds., Mathematical Research, Berlin, 1995, Akademie Verlag.
[6] ——, *Rigorous Global Search: Continuous Problems*, Kluwer, Dordrecht, Netherlands, 1996.
[7] R. Krawczyk and A. Neumaier, *Interval slopes for rational functions and associated centered forms*, SIAM J. Numer. Anal., 22 (1985), pp. 604–616.
[8] A. Neumaier, *Interval Methods for Systems of Equations*, Cambridge University Press, Cambridge, England, 1990.
[9] L. B. Rall, *Automatic Differentiation: Techniques and Applications*, Lecture Notes in Computer Science no. 120, Springer, Berlin, New York, etc., 1981.