An Efficient Digit-by-Digit Decimal Square Root Algorithm Using Non-restoring Pseudo-Division

by C. Bond, ©2003

http://www.crbond.com

Abstract

A square root algorithm optimized for hand held calculators has been previously disclosed in an article by Egbert [1]. The algorithm is similar to other digit-by-digit decimal algorithms published elsewhere, but with a number of improvements to better adapt the method to a class of BCD calculators. In this paper, a further improvement is disclosed which brings the advantages of non-restoring division, also described in the literature, to the pseudo-division process used in taking square roots.

1 BCD Square Root Algorithm

The basic algorithm we will use is completely described in the article by Egbert and the reader is advised to consult the original paper for a detailed explanation.

Egbert shows that the method for computing square roots closely follows the pencil and paper methods taught in school, but includes modifications to reduce the number of extraneous computations. He also shows how to exploit certain properties of the electronic storage medium. For example, a decimal digit can be easily multiplied by ten by simply shifting it to the adjacent position.

1.1 Pseudo-division

The central computation in the algorithm involves calculating each result digit successively, from highest to lowest. The method used for each digit is the same, and requires subtracting an iteratively modified quantity from a suitable starting value until underflow occurs.[1] This process resembles division, except for the repeated updates to the subtractor, and is referred to as *pseudo-division* in the literature.

When underflow occurs, according to Egbert, the last subtractor is added back in, thus restoring the least positive value of the result for the next operation. Note that when this strategy is applied to

[1] Egbert refers to *underflow* as *overdraft* in the article.

ordinary division, it is called restoring division. Hence, in the square root algorithm we call it restoring pseudo-division.

Observe that the goal in these repeated subtractions is to reduce the starting quantity to zero, or as near as possible to zero, from the positive side of the number line. This is analogous to the asymptotic convergence of a continuous function toward zero from one side, but differs in the repeated removal of fixed quantities which are taken from a set of progressively smaller values.

1.2 Non-restoring Pseudo-division

Non-restoring methods converge to zero from both sides alternately. Subtraction is iterated until underflow occurs, after which addition of new, smaller values is repeated until overflow occurs. The process successively reduces the magnitude of the starting value to zero from one side and then from the other.

Although non-restoring processes are more complex, since they must control the algorithm progress from two directions instead of one, they do reduce the total number of calculations by elimination the restoring operation.

2 The Improved Algorithm

The method used by HP in its calculators, as described by Egbert, is explained by an example in the previously cited article. At some point in the algorithm, Egbert shows several consecutive steps in the recovery of one digit, as follows,

$$10a \times 10^j + 05 \times 10^{2j}$$
$$10a \times 10^j + 15 \times 10^{2j}$$
$$10a \times 10^j + 25 \times 10^{2j}$$

where the a value is the current root estimate.

Note that the above expressions are the iteratively modified subtractors which, at some point, will reduce the quantity from which they are subtracted to zero or below it.

In the HP algorithm, underflow is followed by a restore operation. In ours we present the appropriate modification to eliminate the restore by providing an *add up* iteration to complement the *subtract down* algorithm described by Egbert.

Rather than restoring the previously subtracted value and setting up a new subtractor, we simply set up a new quantity to be used as an additive value. Specifically, the subtraction loops begin by appending

a value of 05 to the current root estimate and increment this digit pair so that it becomes 15, 25, etc. on successive iterations. For reasons which will not be explained here, the subtraction will always terminate at or before the digit pair equals 95.

2.1 The Addition Loop

For the addition loop, we append the value 95 to the current root estimate and decrement this digit pair so that it becomes 85, 75, etc. on successive iterations. The control value which previously underflowed will go positive again before or when the digit pair equals 05. As in the case of the subtraction loop, the iteration will always terminate.[2]

An advantage of this strategy is that the continually improving root estimate does not require any separate count controls to identify the currently found digit. It is automatically placed properly in the result.

3 Conclusion

The author has implemented the method of non-restoring pseudo-division for extracting square roots on a number of platforms, including those using different processors and in extended precision BCD math. It is easy to implement and has proven to be reliable, efficient and stable. It is currently used in the author's XBCD extended precision BCD math package.

References

[1] William E. Egbert, "Personal Calculator Algorithms I: Square Roots," *Hewlett-Packard Journal*, vol. 28, no. 9, May 1977.

[2]The value of the result when underflow occurs is not a signed BCD quantity, but can be handled identically in the restoring and non-restoring algorithms.