ISSN (Online) 2393-8021
ISSN (Print) 2394-1588

International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 2, February 2015

Loss of Significance in Numerical Computing and Polynomial Form

Ashu Vij

Assistant Professor, P.G. Department of Mathematics, D.A.V. College, Amritsar

Abstract: One of the most common and often avoidable peculiarity of using normalized floating point representation of numbers in arithmetic operations is Loss of Significance. Use of normalized floating point representation of numbers in arithmetic operations lead to some unexpected and unfortunate consequences. There is a loss of significance due to subtraction of a number from nearly equal number. In this paper, the loss of significance in various numerical computations including its effect in polynomial form is discussed. Further, the alternatives to minimize its effect on final result are also discussed.

Keywords: normalized floating point, arithmetic operations, loss of significance, polynomial form.

INTRODUCTION

In computing, floating point describes a method of representing an approximation of a real number in a way that can support a wide range of values. The numbers are, in general, represented approximately to a fixed number of significant digits and scaled using an exponent. The base for the scaling is normally 2, 10 or 16. The typical number that can be represented exactly is of the form:
Significant digits × base $^{\text{exponent}}$

In scientific notation, the given number is scaled by a power of 10 so that it lies within a certain range- typically between 1 and 10, with the radix point appearing immediately after the first digit.

A floating-point number consists of:

- A signed (meaning positive or negative) digit string of a given length in a given base (or radix). This digit string is referred to as the significand, coefficient or the mantissa . The length of the significand determines the precision to which numbers can be represented. The radix point position is assumed to always be somewhere within the significand—often just after or just before the most significant digit, or to the right of the rightmost (least significant) digit

- A signed integer exponent, also referred to as the characteristic or scale, which modifies the magnitude of the number.

Leonardo Torres y Quevedo in 1914 designed an electro-mechanical version of the Analytical Engine of Charles Babbage which included floating-point arithmetic.[2] In 1938, Konrad Zuse of Berlin completed the Z1, the first mechanical binary programmable computer; it was, however, unreliable in operation.[3] It worked with 24-bit binary floating-point numbers having a 7-bit signed exponent, a 16-bit significand (including one implicit bit), and a sign bit. The memory used sliding metal parts to store 64 words of such numbers. The more reliable relay-based Z3, completed in 1941 had representations for plus and minus infinity. It implemented defined operations with infinity such as $1/\infty = 0$ and stopped on undefined operations like $0\times\infty$. It also implemented the square root operation in hardware.Konrad Zuse, architect of the Z3computer, which used 22-bit binary floating point.

Loss of Significance in numerical computation

The fact that floating-point numbers cannot precisely represent all real numbers, and that floating-point operations cannot precisely represent true arithmetic operations, leads to many surprising situations. This is related to the finite precision with which computers generally represent numbers. For example, the non-representability of 0.1 and 0.01 (in binary) means that the result of attempting to square 0.1 is neither 0.01 nor the representable number closest to it. In 24-bit (single precision) representation, 0.1 (decimal) was given previously as

$e = -4$; $s = 110011001100110011001101$, which is 0.100000001490116119384765625 exactly. Squaring this number gives
0.010000000298023226097399174250313080847263336181640625 exactly.

Squaring it with single-precision floating-point hardware (with rounding) gives 0.010000000707805156707763671875 exactly. But the representable number closest to 0.01 is 0.009999999776482582092285156250 exactly. So we can see that using normalized floating point representation of numbers in arithmetic operations lead to some unexpected and unfortunate consequences. One of the most common and often avoidable peculiarity of using normalized floating point representation of numbers in arithmetic operations is Loss of Significance. In this paper, the loss of significance in various numerical computations including its effect in polynomial form is discussed. Further, the alternatives to minimize its effect on final result are also discussed.

Subtraction of a number from nearly equal number may result in loss of significance or cancellation error. To understand this loss, let us consider the numbers a= .6439E2 and b=.6398E2 each correct to four significant digits.

ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
Vol. 2, Issue 2, February 2015

Now the difference a-b = .6439E2 - .6398E2 = .0041E2, which is correct to only two significant digits.

This loss of significant figures in the subtraction of two nearly equal numbers is greatest source of inaccuracy in most computations and sometimes make the result of computations worthless. This inaccuracy due to loss of significant figures can be lessened and sometimes can be avoided entirely as follows:

1. We should approximate each number involved in subtraction operation with sufficient accuracy before subtraction operation. But this method does not work when two given numbers are known to be true only to a few significant digits.
2. We should transform the expression whose value is desired into another expression so that the subtraction operation can be avoided.

Foe example: To evaluate f(x)= 1-cosx for x=0.05, we have,
f(0.05)=1-cos0.05= .1000E1-.9988E0 =.1000E1 - .0999E1 =.0001E1
Although 1 is exact and cos0.05 is correct to four significant digits but f(0.05) is correct to only one significant digit. To avoid this loss of significance, we proceed as follows:
$1-\cos x = (1-\cos x)(1+\cos x)/(1+\cos x) = \sin^2 x/(1+\cos x)$
It can be verified that more accurate approximation is obtained for above calculation .

Loss of significance in polynomial forms
In all areas of numerical analysis, polynomials are used as the basic means of approximation. So the representation and evaluation of polynomials is a basic topic in numerical analysis. A customary way to represent a polynomial is as follows:

$F(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$
with certain coefficients $a_0, a_1, a_2, \dots\dots$

This power form of polynomial is a standard way to specify a polynomial. But this form may lead to loss of significance[4].

For example: if we construct power form of a straight line which take the values f(1000)= 1/3 and f(1001)= -2/3 then power form for this polynomial is f(x)=1000.3-x in five digit floating point arithmetic.

But f(1000)=.3 and f(1001)= -0.7 using the polynomial form using same five digit floating point arithmetic. This lead to loss of significant digits as we can recover only one decimal digit.

To avoid this loss of significant digits , we use shifted power form as follows:
We change the centre at 1000 then
f(x)= 0.33333-(x-1000)

Using same five digit floating point arithmetic, f(1000)=.33333 and f(1001)= -0.66667

Similarly Chebchev's polynomial or orthogonal polynomial approximation can also be used as a remedy for loss of significance.

CONCLUSION

One should take due care while dealing with floating point numbers in normalized form on computer. The fact that floating-point numbers cannot precisely represent all real numbers, and that floating-point operations cannot precisely represent true arithmetic operations, leads to many surprising situations. This is related to the finite precision with which computers generally represent numbers. There is an incident to see how things can go wrong while using floating point numbers. On 25 February 1991, a loss of significance in a MIM-104 Patriot missile battery prevented it intercepting an incoming Scud missile in Dhahran, Saudi Arabia, contributing to the death of 28 soldiers from the U.S. Army's 14th Quartermaster Detachment[5]

REFERENCES

1. Scarborough,j.b.: Numerical mathematical analysis,1958
2. B. Randell (1982). From analytical engine to electronic digital computer: the contributions of Ludgate, Torres, and Bush. IEEE Annals of the History of Computing, 04(4). pp. 327–341.
3. "Konrad Zuse's Legacy: The Architecture of the Z1 and Z3" . IEEE Annals of the History of Computing 19 (2): 5–15. 1997. doi:10.1109/85.586067 .
4. Elementary numerical analysis: an algorithmic approach : Samuel D. Conte/ Carl de Boor
5. "Patriot missile defense, Software problem led to system failure at Dharhan, Saudi Arabia; GAO report IMTEC 92-26" . US Government Accounting Office.