

Section 1.4 Mathematics on the Computer: Floating Point Arithmetic

Numerical analysis uses algorithms to generate approximate solutions to models of real-world problems and hence **a sequence of calculations is performed usually on a computer. As each floating point operation is performed, a new amount of roundoff error is introduced into the value computed.** It is therefore reasonable to expect that there will be an accumulation of roundoff error as calculations proceed. Sometimes this accumulation is benign, positive and negative deviations tending to maintain a level of error or even a decrease. However, in some instances their can be a significant build up of roundoff error that can dramatically contribute to inaccurate computed values. Here we consider such an issue and provide examples of situations that can cause such behavior.

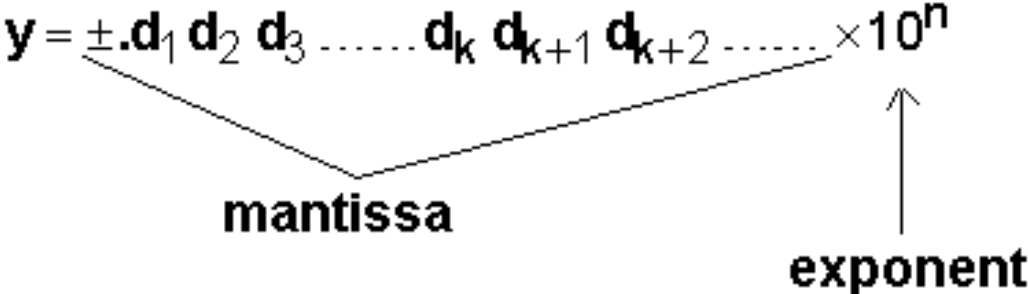
Computers use a model of real arithmetic commonly referred to a **floating point arithmetic**. Floating point arithmetic is a number system that contains a **finite number of elements** each of which is expressible in a **finite number of digits**. So a floating point system is a model for the real number system. In today's digital computers the arithmetic is done in base 2 (binary) and usually adheres to the so-called **IEEE standard**, which is often characterized as a kind of "extended floating point" number system.

What follows is an over view of floating point arithmetic using **base 10** that illustrates general principles of floating point arithmetic systems. Every real number **y** has a decimal representation of the form

$$y = \pm .d_1 d_2 d_3 \dots d_k d_{k+1} d_{k+2} \dots \times 10^n$$

where $1 \leq d_1 \leq 9$ and $0 \leq d_i \leq 9$ for $i > 1$

This referred to as a **normalized decimal form**. It is made up of two parts



A floating point representation of a real number **y** is denoted **fl(y)** and is obtained from the normalized decimal form either by **chopping or rounding after a specified number of digits**.

In a k-digit floating point system we have $fl(y) = \pm .d_1 d_2 d_3 \dots d_k \times 10^n$

where the exponent **n** lies in a particular range like $m \leq n \leq M$.

Example: In MATLAB a **16** digit floating point system is used with an exponent range of **-308 to 308**.

Properties of Floating Point Systems

- Any data entering the system is immediately converted to have the appropriate number of digits (as defined by the system) using the characteristics of the system (chopping or rounding) and checked to see if it is a “valid” value for the system.
- There is a **largest value** in the system.
- There is a **smallest value** in the system.
- If x is a real number then in the system it is $fl(x)$. Any loss of accuracy (significance) is called **ROUND OFF ERROR** regardless whether the system uses chopping or rounding. Also the loss of any accuracy as a result of numerical operations is called **ROUND OFF ERROR**.
- The number zero has a special representation as $+.000\dots0 \times 10^m$.
- Floating point arithmetic is **not associative**; $fl(x) + (fl(y) + fl(z))$ need not be the same as $(fl(x) + fl(y)) + fl(z)$.
- Floating point arithmetic is **not distributive**; $fl(x) \times (fl(y) + fl(z))$ need not be the same as $(fl(x) \times fl(y)) + (fl(x) \times fl(z))$.

Note: Technically the preceding operations of $+$ and \times should be designated by other symbols to denote that is it the floating point arithmetic of addition and multiplication respectively.

Properties of Floating Point Systems (continued)

- Floating point arithmetic operations follow a general procedure like,

→ the value $fl(x)$ is retrieved from its storage location

→ the value $fl(y)$ is retrieved from its storage location

$fl(x)$ and $fl(y)$ together with the “**arithmetic operator**” are used to generate a “result” whose value is placed in a temporary storage area called a **register** or **accumulator** which can retain more digits than the standard storage location

→ **the contents of the register are then converted to a floating point number and placed in a standard storage location; this step often encounters loss of accuracy, that is, roundoff error; this behavior occurs for every arithmetic operation, not at the end of a sequence of operations.**

How does roundoff occur in floating point operations? In the text P.43 is a discussion of this using a general notation. Here we will use a particular operation, namely floating point addition, designated by the symbol \oplus .

- **x and y are real numbers**
- **$fl(x)$ and $fl(y)$ are the floating point representations of x and y respectively**
- **$x + y$ is the value of the sum using real arithmetic**
- **$fl(fl(x) \oplus fl(y))$ is the value of the sum of the floating point representations of x and y in the floating point system**
- **the difference between the floating point value of the sum and the real sum is given by**
 $fl(fl(x) \oplus fl(y)) - (x + y)$

- using algebra we show how to break this into two components and reveal how roundoff error enters

$$fl(fl(x) \oplus fl(y)) - (x + y) = fl(fl(x) \oplus fl(y)) - [fl(x) \oplus fl(y)] + [fl(x) \oplus fl(y)] - (x + y)$$

we added zero in disguise

the term $fl(fl(x) \oplus fl(y)) - [fl(x) \oplus fl(y)]$ is the error incurred when the sum is converted to a floating point number in the system; this is called **introduced error** and is small based on the relative machine accuracy

the term $[fl(x) \oplus fl(y)] - (x + y)$ is the error incurred by approximating the real sum by the sum obtained floating point arithmetic sum; this is called the **propagated error** and can be large

- It can be shown that when using **multiplication and division**, the **relative error propagates slowly** in the sense that, if the relative error in the data is small, the relative error in the product or quotient will also be small. It should be noted, though, that the **absolute error can grow rapidly when multiplying by a large number or dividing by a small number**. Whenever possible algorithms should therefore be written to avoid these situations.
- When working with **addition and subtraction**, the **behavior of the absolute and relative error is very different**. When two nearly equal numbers are subtracted or when numbers of nearly equal magnitude but different signs are added drastic propagation of error can occur; this is called **cancellation error** or **subtractive cancellation error**. Such situations should be avoided.
- Another situation that arises is the addition of numbers of the same sign, but of widely different magnitudes. In this case the impact of the smaller number on the sum can completely disappear.

PITFALLS of FLOATING POINT COMPUTATION (Summary)

There are several types of floating point arithmetic expressions that are to be avoided if possible. These expressions are considered sources of roundoff error; that is, instances in which loss of significance may occur. **About the best we can do is recognize these situations and try to restructure computations to avoid them.**

Addition of numbers of widely varying magnitude.

Subtraction of nearly equal values.

Division by small values or equivalently multiplication by large values.

EXAMPLE We are in a 3-digit floating point system that uses chopping. So if $y = \pm.d_1 d_2 d_3 \dots d_k d_{k+1} d_{k+2} \dots \times 10^n$ is a real number in normalized form, then

$$\text{fl}(y) = \pm.d_1 d_2 d_* \times 10^n$$

where the last digit is d_3 since we chop. (Had the system rounded, then the last digit could have been d_3 or $d_3 + 1$ if $d_4 > 4$.)

Given the real numbers $x = \frac{2}{3}$, $y = \frac{5}{9}$, $z = 983$ we have that

$$\text{fl}(x) = 0.666, \text{fl}(y) = 0.555, \text{fl}(z) = 0.983 \times 10^3.$$

a) Compute $x \oplus y$

We assume the values are taken into the system so we will be adding $\text{fl}(x)$ and $\text{fl}(y)$ within the system. This produces

$$\text{fl}(\text{fl}(x) + \text{fl}(y)) = \text{fl}(0.666 + 0.555) = \text{fl}(1.221) = 0.122 \times 10^1$$

This addition's result is placed in a register & has value 1.221

Note the loss of accuracy in this result.

b) Compute $x \otimes y$

We assume the values are taken into the system so we will be multiplying $fl(x)$ and $fl(y)$ within the system. This produces

$$fl(fl(x) \times fl(y)) = fl(0.666 \times 0.555) = fl(0.36963) = 0.369$$

This product's result is placed in a register & has value 0.36963

Note the loss of accuracy in this result.

c) Compute $x \oplus z$

We assume the values are taken into the system so we will be adding $fl(x)$ and $fl(z)$ within the system. This produces

$$fl(fl(x) + fl(z)) = fl(0.666 + 0.983 \times 10^3) = fl(983.666) = 0.983 \times 10^3$$

This addition's result is placed in a register & has value 983.666

Note the loss of accuracy in this result.

In effect in this floating point addition x played the role of zero.

Example: Suppose we are in a 7 digit floating point system and we have two real numbers **x** and **y** whose floating point values are accurate to all 7 digits shown:

$$\text{fl}(x) = 0.5491234 \times 10^3 \quad \text{fl}(y) = 0.5490012 \times 10^3$$

Then $\text{fl}(\text{fl}(x) - \text{fl}(y)) = 0.0001222 \times 10^3 = 0.1222\text{efg} \times 10^0$ where digits e, f, and g are set to zero in IEEE systems. **The accuracy of the result is just 4 digits.** So if this result is used in further computations we can only claim that the succeeding values are accurate to at most 4 digits, not 7.

Example: Suppose number **z** has a floating point representation that can be expressed as

$$\text{fl}(z) = z + \text{error} = z + \delta.$$

Divide **z** by $\varepsilon \neq 0$. We get $\frac{z}{\varepsilon} \approx \text{fl}\left(\frac{\text{fl}(z)}{\text{fl}(\varepsilon)}\right) = \text{fl}\left(\frac{z + \delta}{\text{fl}(\varepsilon)}\right)$. Suppose that $\varepsilon = 10^{-n}$, $n > 0$, then

$$\text{Exact value} = \frac{z}{\varepsilon} = z \times 10^n$$

$$\text{Floating Point Value} = \text{fl}\left(\frac{z + \delta}{\text{fl}(\varepsilon)}\right) = (z + \delta) \times 10^n$$

So 10^n multiplies the error δ , hence there error is magnified.

Example: Structuring computations to avoid a pitfall of computation.

The midpoint of an interval $[a, b]$ is usually computed as

$$\text{mid}_1 = .5 \times (a + b) \quad (1)$$

An alternate way to compute the midpoint of $[a, b]$ is

$$\text{mid}_2 = a + .5 \times (b - a) \quad (2)$$

Using exact arithmetic the results from (1) and (2) are identical. However, in floating point arithmetic there may be a significant difference in the output from these two formulas.

Let $a = 0.982$ and $b = 0.987$. Assume that we are using 3 digit chopping arithmetic. From (1) we have

$$\text{mid}_1 = \text{fl}(.5 \times (\text{fl}(a) + \text{fl}(b))) = .5 \times \text{fl}(1.969) = .5 \times 1.96 = 0.980$$

and from (2) we get

$$\begin{aligned} \text{mid}_2 &= \text{fl}(a) + \text{fl}(.5 \times (\text{fl}(\text{fl}(b)) - \text{fl}(a))) = 0.982 + \text{fl}(.5 \times \text{fl}(0.987 - 0.982)) \\ &= 0.982 + \text{fl}(.5 \times 0.005) = 0.982 + \text{fl}(0.0025) = \text{fl}(0.9845) = 0.984 \end{aligned}$$

What is the distinguishing difference between the two outputs, other than they are different?

Example: Roundoff Error in Function Evaluation

Function $f(x) = \frac{1 - \cos(x)}{x^2}$ near $x = 0$ will experience difficulties in evaluation.

Rewriting the expression as $g(x) = \left(\frac{\sin(x)}{x}\right)^2 \left(\frac{1}{1 + \cos(x)}\right)$ improves the situation.

x	$(1 - \cos(x))/x^2$	$(\sin(x)/x)^2 (1/(1+\cos(x)))$
1.0000000000000000e-002	4.999958333473664e-001	4.999958333472223e-001
1.0000000000000000e-003	4.999999583255033e-001	4.999999583333346e-001
1.0000000000000000e-004	4.999999969612645e-001	4.99999995833334e-001
1.0000000000000000e-005	5.000000413701854e-001	4.99999999958332e-001
1.0000000000000000e-006	5.000444502911705e-001	4.99999999999584e-001
9.999999999999998e-008	4.996003610813205e-001	4.99999999999996e-001
1.0000000000000000e-008	0	5.000000000000000e-001

<Calculations done in MATLAB.>

Example: Impact of Floating Point Arithmetic on a Linear System

Consider the solution of the linear system of three equations in three unknowns, given in augmented matrix form by

$$\left[\begin{array}{ccc|c} 6 & -2 & 3 & 5 \\ 1 & -1/3 & 1/3 & 2 \\ 1 & 3 & -1 & 5 \end{array} \right].$$

Here we will use row operations to reduce the augmented matrix to upper triangular form and then use back substitution. Performing row operations to “zero out” the 2nd and 3rd entries in column 1 using exact arithmetic we get

$$\left[\begin{array}{ccc|c} 6 & -2 & 3 & 5 \\ 0 & 0 & -1/6 & 7/6 \\ 0 & 10/3 & -3/2 & 25/6 \end{array} \right],$$

from which the exact solution is computed to be: $x_1 = 3.7$, $x_2 = -1.9$ and $x_3 = -7$. To arrive at these values, interchange the second and third rows of the reduced matrix and perform back substitution.

To examine the effect of working in 4 decimal rounding arithmetic we first compute the matrix entries as

$$\left[\begin{array}{ccc|c} 6 & -2 & 3 & 5 \\ 1 & -0.3333 & 0.3333 & 2 \\ 1 & 3 & -1 & 5 \end{array} \right]$$

Next we determine $\text{fl}(-1/6) = -0.1667$ and perform the row operations to “zero out” the 2nd and 3rd entries in column 1 using 4 decimal rounding arithmetic; we get

$$\left[\begin{array}{ccc|c} 6 & -2 & 3 & 5 \\ 0 & 0.0001 & -0.1668 & 1.167 \\ 0 & 3.333 & -1.500 & 4.167 \end{array} \right]$$

The details of computations for the entries in the second row are as follows:

$$-0.3333 + (-2) \times (-0.1667) = -0.3333 + 0.3334 = \underline{0.0001};$$

$$0.3333 + (3) \times (-0.1667) = 0.3333 - 0.5001 = \underline{-0.1668}; \text{ and}$$

$$2 + (5) \times (-0.1667) = 2 - 0.8335 = \underline{1.167}.$$

The calculations for the third row are shown next:

$$3 + (-2) \times (-0.1667) = 3 + 0.3334 = \underline{3.333};$$

$$-1 + (3) \times (-0.1667) = -1 - 0.5001 = \underline{-1.500}; \text{ and}$$

$$5 + (5) \times (-0.1667) = 5 - 0.8335 = \underline{4.167}.$$

Observe that when using 4 digit rounding arithmetic the (2,2)-entry is not zero.

$$\left[\begin{array}{ccc|c} 6 & -2 & 3 & 5 \\ 0 & \underline{0.0001} & -0.1668 & 1.167 \\ 0 & 3.333 & -1.500 & 4.167 \end{array} \right]$$

Thus to obtain upper triangular form we need to perform another row operation to eliminate the (3,2)-entry. When we do this we need to compute $\text{fl}(-3.333/0.0001)$ which gives the value -33330 . Using this value to obtain upper triangular form we perform the following 4 digit arithmetic:

$$\begin{aligned} -1.500 + (-0.1668) \times (-33330) &= -1.500 + 5559 = 5558; \text{ and} \\ 4.167 + (1.167) \times (-33330) &= 4.167 - 38900 = -38900. \end{aligned}$$

The resulting matrix is

$$\left[\begin{array}{ccc|c} 6 & -2 & 3 & 5 \\ 0 & 0.0001 & -0.1668 & 1.167 \\ 0 & 0 & 5558 & -38900 \end{array} \right].$$

$$\left[\begin{array}{ccc|c} 6 & -2 & 3 & 5 \\ 0 & 0.0001 & -0.1668 & 1.167 \\ 0 & 0 & 5558 & -38900 \end{array} \right].$$

Now using back substitution we have the following.

From the last row, it follows that $x_3 = -6.999$, which is a pretty good approximation for the exact value of x_3 . Substituting $x_3 = -6.999$ into the second row generates the equation $0.0001x_2 - 0.1668(-6.999) = 1.167$, which is equivalent to $0.0001x_2 + 1.167 = 1.167$ in 4 decimal digit rounding arithmetic. The solution of this last equation is $x_2 = 0.0000$. Finally, substituting the values for x_2 and x_3 into the first row gives the equation $6x_1 + (-6.999) = 5$, whose solution is $x_1 = 4.333$. The error in x_2 is 100%, while that in x_1 is slightly more than 17%. These large errors can be directly traced to the loss of significant figures caused ultimately by the subtraction of nearly equal numbers.

Recall this computation for entries in the second row.

$$-0.3333 + (-2) \times (-0.1667) = -0.3333 + 0.3334 = 0.0001;$$

This subtraction of equal values is directly related to computing $\text{fl}(-3.333/0.0001)$ which involved dividing by a small number.