Chapter 2

Floating-point Computation

2.1 Positional Number System

An integer $N$ in a number system of base (or radix) $\beta$ may be written as

$$N = a_n \beta^n + a_{n-1} \beta^{n-1} + \cdots + a_1 \beta + a_0 = P_n(\beta)$$

where $a_i$ are positive integers less than $\beta$ and $P_n(\beta)$ is a polynomial of degree $n$. For example, in the decimal system:

$$\beta = 10, \quad 0 \leq a_i \leq 9$$

while in the binary system

$$\beta = 2, \quad 0 \leq a_i \leq 1.$$  

A fraction $z$ in base $\beta$ may be written as

$$z = a_{-1} \beta^{-1} + a_{-2} \beta^{-2} + \cdots + a_{-k} \beta^{-k} + \cdots = \sum_{j=1}^{\infty} a_{-j} \beta^{-j}$$

Note that the above series always converges. For example, if all $a_{-j} = 1$, then, for $\beta = 2$

$$z = \sum_{j=1}^{\infty} 2^{-j} = \frac{1/2}{1 - 1/2} = 1.$$  

Recall that the sum of a geometric series with $n$ terms is given by

$$a + ar + ar^2 + ar^3 + \cdots + ar^{n-1} = a \left( \frac{1 - r^n}{1 - r} \right).$$

Also, the geometric series with infinite terms, i.e., $n = \infty$,

$$a + ar + ar^2 + ar^3 + \cdots = \frac{a}{1 - r}$$

provided $-1 < r < 1$.

**Example 2.1** Convert the binary integer $N = (110100010)_2$ to decimal.

**Solution** Before we compute $N$, we first describe an efficient way to compute the value of a polynomial. As an example, we can rewrite a degree 3 polynomial as follows

$$P_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

$$= a_0 + x (a_1 + x(a_2 + x a_3))$$

and evaluate the terms in the order of the parenthesis. This is equivalent to the Horner’s rule for evaluating polynomials.
Horner's rule for polynomials

\[ \begin{align*}
  b_0 &= a_n \\
  \text{for } k &= 1, 2, \ldots, n \\
  b_k &= a_{n-k} + b_{k-1} \times x \\
\end{align*} \]
end;

Operation Count: \(2n\) arithmetic steps

\[
\begin{array}{ccccccccccc}
  k &= 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
  a_{-k} &= 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
  b_k &= 1 \rightarrow 3 \rightarrow 6 \rightarrow 13 \rightarrow 26 \rightarrow 52 \rightarrow 104 \rightarrow 209 \rightarrow 418
\end{array}
\]

Figure 2.1: Converting binary integer \(110100010_2\) to decimal.

Therefore, \(N = (110100010)_2 = (418)_{10}. \)

Example 2.2 Convert the decimal integer \(N = (418)_{10}\) to binary.

Solution To convert from decimal to binary, we make use of the fact that if

\[ N = a_n \times 2^n + a_{n-1} \times 2^{n-1} + \cdots + a_1 \times 2 + a_0, \]

then for

\[
\begin{align*}
  N : \text{even} & \Rightarrow a_0 = 0 \\
  N : \text{odd} & \Rightarrow a_0 = 1.
\end{align*}
\]

Using this fact, we get

\[
\begin{align*}
  N_0 &= 418 \\
  N_1 &= \frac{N_0 - a_0}{2} = 209 \\
  N_2 &= \frac{N_1 - a_1}{2} = 104 \\
\end{align*}
\]

\[
\begin{array}{ccccccccccc}
  k &= 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
  a_k &= 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\
  N_k &= 418 & 209 & 104 & 52 & 26 & 13 & 6 & 3 & 0 \\
\end{array}
\]

Figure 2.2: Converting decimal integer \((418)_2\) to binary.

Therefore, \(N = (418)_{10} = (110100010)_2. \)

Example 2.3 Let us turn our attention to decimal fractions. It is important to note that not all decimal fractions are exactly representable in the binary system. As an example, convert \(z = (0.1)_{10}\) to binary.

Solution The algorithm for converting a decimal fraction to binary is
Converting decimal fraction to binary

\[ z_1 = z \]
for \( k = 1, 2, 3, \ldots \)
\[ a_{-k} = \begin{cases} 1 & \text{if } 2z_k \geq 1 \\ 0 & \text{if } 2z_k < 1 \end{cases} \]
\[ z_{k+1} = 2z_k - a_{-k} \]
end;

Let
\[ z_1 = (0.1)_{10} = a_{-1}2^{-1} + a_{-2}2^{-2} + a_{-3}2^{-3} + \cdots \]

Therefore,
\[ 2z_1 = a_{-1} + a_{-2}2^{-1} + a_{-3}2^{-2} + \cdots \]

Following the algorithm,
\[
\begin{align*}
z_1 &= (0.1)_{10}; \\
z_2 &= 2z_1 - a_{-1} = 0.2; \\
z_3 &= 2z_2 - a_{-2} = 0.2; \\
z_4 &= 2z_3 - a_{-3} = 0.2; \\
z_5 &= 2z_4 - a_{-4} = 0.2; \\
z_6 &= 2z_5 - a_{-5} = 0.2; \\
\end{align*}
\]

Complete this example to show that
\[ z = (0.1)_{10} = (0.0001100110011\cdots)_2, \]
i.e., \( z \) has a nonterminating binary representation! \( \square \)

Terminating an infinite binary fraction. Suppose we terminate an infinite binary fraction after \( t \) digits, and convert to decimal. The magnitude of the maximum error cannot exceed
\[
\begin{align*}
y &= 2^{-(t+1)} + 2^{-(t+2)} + 2^{-(t+3)} + \cdots \\
&= 2^{-(t+1)}[1 + 2^{-1} + 2^{-2} + \cdots] \\
&= 2^{-(t+1)} \left( \frac{1}{1 - \frac{1}{2}} \right) = 2^{-t}.
\end{align*}
\]

2.2 Floating-point Arithmetic

Scientific calculations are carried out in floating-point arithmetic. This is a number system which uses a finite number of digits to approximate the real number system we use in exact computation.

Let \( x \neq 0 \) be in the real number system,
\[ x = \pm 10^e(0.d_1d_2d_3\cdots), \]
\[ d_1 \neq 0, \quad 0 \leq d_j \leq 9, \quad j > 1. \]

The \( t \)-digit floating-point (decimal) representation of \( x \) is given by
\[ fl(x) = \pm 10^e(0.\delta_1\delta_2\cdots\delta_t), \]
\[ \delta_1 \neq 0, \quad 0 \leq \delta_j \leq 9, \quad j \geq 2 \]

The condition \( \delta_1 \neq 0 \) implies that it is a normalized floating-point number. Here, \( e \) is called the exponent, and \( (\delta_1\delta_2\cdots\delta_t) \) is called the mantissa or fraction. Usually,
\[ -N \leq e \leq M, \]
and
\[ e < -N \quad \Rightarrow \text{underflow} \]
\[ e > M \quad \Rightarrow \text{overflow} \]

The floating-point numbers are not uniformly distributed, unlike the mathematician’s real number system, but are clustered around zero. This can be seen from the following examples for decimal and binary systems.

**Example 2.4** \( \beta = 10; \ t = 3; \ N = M = 3. \)

Aside from zero, the numbers are
\[
\begin{align*}
\pm 0.100 \times 10^{-3} \\
\pm 0.101 \times 10^{-3} \\
\vdots \\
\pm 0.999 \times 10^{-3} \\
\pm 0.100 \times 10^{-2} \\
\pm 0.101 \times 10^{-2} \\
\vdots \\
\pm 0.999 \times 10^{-2} \\
\pm 0.100 \times 10^{+3} \\
\pm 0.101 \times 10^{+3} \\
\vdots \\
\pm 0.999 \times 10^{+3}
\end{align*}
\]

Spacing = \(0.001 \times 10^{-3} = 10^{-6}\)

\(\square\)

**Example 2.5** \( \beta = 2; \ t = 3; \ N = 1; \ M = 2. \)

Aside from zero, the numbers are:
\[
\begin{align*}
\pm 0.100 \times 2^{-1} \\
\pm 0.101 \times 2^{-1} \\
\pm 0.110 \times 2^{-1} \\
\pm 0.111 \times 2^{-1} \\
\pm 0.100 \times 2^{+0} \\
\pm 0.101 \times 2^{+0} \\
\pm 0.110 \times 2^{+0} \\
\pm 0.111 \times 2^{+0} \\
\vdots \\
\pm 0.100 \times 2^{+2} \\
\pm 0.101 \times 2^{+2} \\
\pm 0.110 \times 2^{+2} \\
\pm 0.111 \times 2^{+2}
\end{align*}
\]

Spacing = \((0.001)_2 \times 1 = \frac{1}{16}\)

\(\square\)

In general, for a floating point system with radix \(\beta\) and mantissa of length \(t\)
\[ fl(x) = \pm \beta^e (0.\delta_1 \delta_2 \cdots \delta_t) \quad \delta_1 \neq 0; \quad 0 \leq \delta_1 \leq \beta - 1, \]
floating point numbers in a given group are given by

\[
\begin{array}{c|c}
\pm & 0.100 \cdots 0 0 \cdot \beta^e \\
\pm & 0.100 \cdots 0 1 \cdot \beta^e \\
\vdots & \\
\pm & 0.vvv \cdots v v \cdot \beta^e \\
\pm & 0.100 \cdots 0 0 \cdot \beta^{e+1} \\
v & = \beta - 1
\end{array}
\]

spacing = \gamma = \beta^e \cdot \beta^{-t} = \beta^{e-t}

Also, the number of floating point numbers in a group is given by

\[
\frac{\beta^e - \beta^{e-1}}{\beta^{e-t}} = \frac{\beta - 1}{\beta^{1-t}}
\]

Therefore, total number of floating point numbers in the system is given by

\[
2 \left( \frac{\beta - 1}{\beta^{1-t}} \right) (M + N + 1) + 1.
\]

Also, the floating point number with smallest magnitude is \( \pm \beta^{-N-1} \) whereas the floating point number with largest magnitude is \( \beta^M - \beta^{M-t} \).

Next, we describe how to obtain floating point digits from the exact representation of a number. Consider

\[ x = \pm \beta^e (0.d_1d_2d_3 \cdots), \quad d_1 \neq 0; \quad 0 \leq d_j \leq \beta - 1, \]

i.e., \( \beta^{e-1} \leq x < \beta^e \). We present only two ways in which the floating point digits \( d_i \) are obtained from the exact digits \( d_i \).

**Chopping.**

\[ \delta_i = d_i \quad i = 1, 2, \ldots, t \]

Observing that, in the interval \([\beta^{e-1}, \beta^e]\), the floating point numbers are uniformly distributed with a separation of \( \beta^{e-t} \), we see that

\[ |fl(x) - x| \leq \beta^{e-t} \]

and

\[ \left\| \frac{fl(x) - x}{x} \right\| \leq \frac{\beta^{e-t}}{|x|} \leq \frac{\beta^{e-t}}{\beta^{e-1}} = \beta^{1-t}, \]

i.e.,

\[ fl(x) = x(1 + \mu_c), \]

where \( |\mu_c| \leq \beta^{1-t} \).
Rounding.

\[(\delta_1\delta_2\cdots\delta_t) = \left\lfloor d_1d_2\cdots d_t \cdot d_{t+1} + \frac{1}{2} \right\rfloor\]

where \(|y|\) is the greatest integer less than or equal to \(y\). Therefore,

\[|f(x) - x| \leq \frac{1}{2} \beta^{c-t}\]

or

\[\left| \frac{f(x) - x}{x} \right| \leq \frac{1}{2} \beta^{c-t-1} = \frac{1}{2} \beta^{1-t}\]

where \(|\mu_R| \leq \frac{1}{2} \beta^{1-t} \).

\\

\[
\begin{align*}
f(x) &= x(1 + \delta); \quad |\delta| \leq \epsilon \quad [\epsilon = \text{unit roundoff}] \\
\epsilon &= \begin{cases} \\
\beta^{1-t} & \text{chopping} \\
\frac{1}{2} \beta^{1-t} & \text{rounding}
\end{cases}
\end{align*}
\]

We assume throughout our error analysis of basic arithmetic operations that our hypothetical computer performs each arithmetic operation correctly to 32 digits. Hence, for two normalized numbers

\[
a = 0 \cdot a_1a_2 \cdots a_t \cdot \beta^c \\
b = 0 \cdot b_1b_2 \cdots b_t \cdot \beta^l
\]

we have

\[f(x) = x(1 + \delta)\]

where \(\bullet\) represents operations such as \(+, \ast, \text{ or } /\), and \(|\delta| \leq \epsilon\) (unit roundoff).

**Example 2.6**

\[
f(x + y + z) = f(f(x + y) + z) = [(x + y)(1 + \delta_1) + z](1 + \delta_2) = (x + y)(1 + \delta_1)(1 + \delta_2) + z(1 + \delta_2)
\]

Therefore,

\[f(x + y + z) \cong (x + y + z) + [(x + y)(\delta_1 + \delta_2) + z \cdot \delta_2]
\]

and

\[|f(x + y + z) - (x + y + z)| \leq 2\epsilon|x + y + z|.
\]

In general,

\[|f\left(\sum_{i=1}^{n} x_i\right) - \left(\sum_{i=1}^{n} x_i\right)| \leq (n - 1)\epsilon\left(\sum_{i=1}^{n} |x_i|\right)
\]

**Example 2.7** The order of computation is important. For example, one may obtain different errors depending on the order in which a set of numbers are added. Let us analyse the error in the following computations.

(i) \(f(x_1 + x_2 + x_3)\) \(x_1 > x_2 > x_3\)

(ii) \(f(x_3 + x_2 + x_1)\) \(x_1 > x_2 > x_3\)

**Solution**
(i) \( z_1 = x_1 \)
\[ z_2 = x_2 + z_1 = [10^4(0.00125400 + 0.38270000)] = f\l[10^4(0.38395400)] = 10^4(0.3840) \]
\[ z_3 = x_3 + z_2 = [10^4(0.38400000 + 0.00015670)] = f\l[10^4(0.38415670)] = 10^4(0.3842) \]

Exact answer := \( 10^4(0.38411070) \), i.e., we have one unit error in the last figure.

(ii) \( z_1 = x_3 \)
\[ z_2 = x_2 + z_1 = [10^2(0.12540000 + 0.01567000)] = f\l[10^2(0.14107000)] = 10^2(0.1411) \]
\[ z_3 = x_1 + z_2 = [10^4(0.38270000 + 0.00141100)] = f\l[10^4(0.38411100)] = 10^4(0.3841), \text{ i.e., correct to 4 decimals.} \]

If error in \( \sum_{i=1}^{n} x_i \) is to be minimized, we should have the numbers of largest magnitude be added last; i.e., associated with the smallest possible error; and the numbers of smallest magnitude added first (i.e., associated with the largest error).

### 2.3 Condition of Problems and Stability of Algorithms

Poor accuracy in the output can depend either on the problem or the algorithm. The problem can be ill-conditioned; i.e., small perturbations in the input data could lead to large perturbations in the output. Or, the algorithm may be poorly constructed, i.e., unstable.

**Example 2.8** Consider the linear system

\[
\begin{align*}
0.913x_1 + 0.659x_2 &= 0.254 \\
0.780x_1 + 0.563x_2 &= 0.217
\end{align*}
\]

whose exact solution is:

\[
\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1.0 \\ -1.0 \end{pmatrix}.
\]

Compute this solution using 3-decimal arithmetic with chopping.

**Solution** Multiply the first equation by

\[
m = -f\l(\frac{0.780}{0.913}) = -f\l(0.854326) = -0.854
\]

and add to the second equation. Thus we get,

\[
0 \cdot x_1 + x_2 \cdot f\l(0.563 - f\l(0.659 \cdot 0.854)]
= f\l(0.217 - f\l(0.254 \cdot 0.854)]
\]
or

\[ [f(l(0.563 - 0.562))]x_2 = f(l(0.217 - 0.216)) \]
\[ 0.001x_2 = 0.001 \]
\[ \text{i.e. } \hat{x}_2 = 1. \]

Substituting in the 1st equation, we get

\[ 0.913x_1 = f(l(0.254 - 0.659)) = -0.405 \]

or

\[ \hat{x}_1 = f(l(-0.405/0.913)) = -0.443. \]

Let us compare the computed solution to the exact solution.

Computed: \( \begin{pmatrix} -0.443 \\ 1 \end{pmatrix} \)

Exact: \( \begin{pmatrix} 1 \\ -1 \end{pmatrix} \)

The computed solution does not even have the correct sign! What went wrong? The disastrous calculation was that of computing \( x_2 \), i.e.,

\[ [f(l(0.563 - 0.562))]x_2 = f(l(0.217 - 0.216)) \]

Clearly, we cancelled all the significant digits and obtained a result heavily contaminated with rounding errors. The situation, however, is readily corrected if one uses 6-decimal arithmetic.

\[ x^2 - 6.433x + 0.009474 = 0 \]

Using 4-decimal arithmetic with rounding.

**Solution** The roots of the quadratic equation

\[ ax^2 + bx + c = 0 \]

are given by

\[ x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]

The smaller root is given by

\[ x = \frac{1}{2} \left[ 6.433 - \sqrt{(6.433)^2 - 4(0.009474)} \right] \]

and the calculation is organized as follows:

1. \( u = f(l[10^1(0.6433)])^2 = 10^2(0.4138) \)
2. \( v = f(l[10^1(0.4000) + 10^{-2}(0.9474)]) = 10^{-1}(0.3790) \)
3. \( w = f(l(u - v)) = f(l[10^2(0.41380000 - 0.00037900)]) = 10^2(0.4134) \)
4. \( y = f(l(\sqrt{w})) = 10^1(0.6430) \)
5. \( z = f(l[10^1(0.6433 - 0.6430)]) = 10^{-2}(0.3000) \)
6. \( \tilde{x} = f(l(z/2.0)) = 10^{-2}(0.1500) \)
The computed value \( \tilde{\alpha} \) fails to agree with the true value \( \alpha = 0.0014731 \) in its second significant digit. Again, the disastrous calculation occurs in step 5 when we subtract nearly equal quantities.

Both examples ?? and ?? appear similar; in both cases catastrophic cancellation results in poor solutions. In spite of the similarity, the two examples fail for quite different reasons. The difficulty in example ?? is caused by a poorly constructed algorithm. This difficulty can be avoided by an alternate expression for the smaller root:

\[
x_+ = \frac{1}{2a} \left( -b - \sqrt{b^2 - 4ac} \right) = \frac{1}{2a} \left( \frac{4ac}{-b + \sqrt{b^2 - 4ac}} \right)
\]

Thus, we compute \( \alpha \) as follows:

\[
\alpha = \frac{1}{2} \left( 6.433 - \sqrt{\cdots} \right) \left( 6.433 + \sqrt{\cdots} \right).
\]

Now,

\[
\hat{\alpha} = \frac{1}{2} \left( 6.433 + \sqrt{(6.433)^2 - 4 \cdot 0.009474} \right) = 10^{-2}(0.1473)
\]

which agrees with the exact solution to 4 significant digits.

On the other hand, there are no alternative rearrangements of the computations in example ?? that will allow us to solve the problem accurately using only 3-digit arithmetic (with chopping).

For example ??, we see that a stable algorithm for computing the roots of a quadratic \( ax^2 + bx + c = 0 \) may be outlined as follows:

\[
\rho_1 = -\frac{b + \text{sign}(b) \sqrt{b^2 - 4ac}}{2a}, \quad \rho_2 = \frac{c}{a \cdot \rho_1}.
\]

Note that \( \rho_1 + \rho_2 = -b/a, \rho_1 \cdot \rho_2 = c/a. \)

### 2.4 Ill-Conditioned Problems

The ill-conditioning of a problem is measured by a number \( K_P \) called the condition number of the problem \( P \). The larger \( K_P \) is the more ill-conditioned is the problem.

Let \( P \) be the problem of computing the value of \( f(x) \) at the point \( x = \alpha \) (solution). Let the relative perturbation to the data be of magnitude \( \frac{\Delta \alpha}{\alpha} \). Then

\[
K_P \geq \left| \frac{\Delta \alpha}{\alpha} \right| = \left| \frac{f(\alpha + \Delta \alpha) - f(\alpha)}{f(\alpha)} \right|.
\]

From Taylor series

\[
f(\alpha + \Delta \alpha) \approx f(\alpha) + \Delta \alpha f'(\alpha) + \frac{(\Delta \alpha)^2}{2!} f''(\alpha) + \cdots
\]

Therefore,

\[
K_P \geq \left| \frac{\Delta \alpha}{\alpha} \right| \approx \left| \frac{\Delta \alpha f'(\alpha)}{f(\alpha)} \right|,
\]
i.e.,

\[ K_P \simeq \left| \frac{\alpha f'(\alpha)}{f(\alpha)} \right|. \]

**Example 2.10** Consider the two equations in \( x \), and \( y \)

\[
\begin{align*}
  x + \beta y &= 1 \\
  \beta x + y &= 0;
\end{align*}
\]

for which the exact solution is given by

\[ x = 1/(1 - \beta^2) \quad \& \quad y = \beta/(\beta^2 - 1). \]

Clearly, \( x \) & \( y \) suffer the consequences of severe cancellation when \( \beta \) is close to 1. The condition number of the problem of determining \( x(\beta) \) alone, is given by

\[ K_P \simeq \left| \frac{\beta x'(\beta)}{x(\beta)} \right| = \left| \beta \cdot \frac{2\beta}{(1 - \beta^2)^2} \cdot (1 - \beta^2) \right| = \left| \frac{2\beta^2}{(1 - \beta^2)} \right|. \]

which becomes large for \( \beta \simeq 1 \). Hence, the problem of determining \( x \) is ill-conditioned for \( \beta \) near 1. On the other hand, the problem for determining

\[ z(\beta) = x + y = 1/(1 + \beta) \]

is well-conditioned for any value of \( \beta > 0 \). In fact,

\[ K_{P'} \simeq \left| \frac{\beta z'(\beta)}{z(\beta)} \right| = \left| \beta \cdot \frac{-1}{(1 + \beta)^2} \cdot (1 + \beta) \right| = \left| \frac{\beta}{(1 + \beta)} \right|, \]

which is much smaller than \( K_P \) for \( \beta \simeq 1 \). \[ \square \]

Figure ?? shows an ill-conditioned problem. Although ill-conditioned problems are intrinsically difficult to solve on computer, one can still design effective algorithms for these problems. A *stable algorithm* for an ill-conditioned problem is an algorithm for which the computed solution is near the exact solution of another slightly perturbed problem. Fig. ?? shows an ill-conditioned problem for which an algorithm computes the solution \( f^*(x) \) at point \( x \). The algorithm is said to be stable since the computed solution is near the exact solution \( f(x^*) \) at point \( x^* \), where \( x^* \) is assumed to a slight perturbation of \( x \).
Figure 2.5: A stable algorithm for ill-conditioned problem generates a computed solution that is close to the solution of a slightly perturbed problem.