Part 2

수치해석 입문

>> 2부에서 다루는 내용

Part 2 들어가기 전에

수치해석 소개

Part 2에서는 수치해석의 기초 내용을 학습할 것이다. 먼저 상식으로 알고 있는 숫자 이야기부터 시작해보자. 하루 일과 중 눈에 가장 많이 들어오는 기호가 무엇일까? 한글, 영어 혹은 그 밖의 언어, 마땅한 대답이 떠오르지 않는다면 0, 1, 2, 3…과 같은 숫자를 정답으로 추천하고 싶다. 아침에 일어나서 잠자리에 들 때까지 온종일 숫자를 보고, 누르고, 확인하면서 보낸다. 숫자를 인류 최고의 발명품 혹은 유일한 세계 공용어라고 칭송하는 사람들도 있다.

오늘날 숫자를 가장 일반적이고 보편적으로 표현하는 기호는 아라비아 숫자 체계다. 서로 다른 기호 10개 '0, 1, 2, 3, 4, 5, 6, 7, 8, 9'를 이용하여 위치적 체계로 표시한다. 예를 들어 3,824의 경우 가장 왼쪽 자리에 있는 숫자 3이 가장 큰 값을 나타낸다.

또 아라비아 숫자 만큼은 아니지만, 학술적인 목차, 연속으로 진행되는 행사 혹은 기념 등의 횟수를 표기하는 데 로마 숫자도 종종 사용한다. 로마 숫자는 영어 문자를 사용한 일정한 법칙에 따라 표기한다. 기본적인 로마 숫자의 체계는 아라비아 숫자 1, 5, 10, 50, 100, 500, 1000에 대응해서 I, V, X, L, C, D, M을 각각 사용한다. 예를 들어 3을 표시해보면 I를 오른쪽에서 연속으로 세 개 이어서 III라고 쓴다. 4는 오른쪽 V에서 왼쪽의 I를 빼는 원리로 IV로 표시한다. 그러면 49를 로마 숫자로 나타내려면 어떻게 해야 할까? 40을 먼저 표시하고 9를 표시하면 40은 50에서 10을 빼는 형태이므로 XL이고, 9는 10에서 1을 빼는 형태이므로 IX가 된다. 이제 왼쪽에서 오른쪽으로 40과 9를 차례대로 표시하면 XLIX이다.

아라비아 숫자와 로마 숫자 중 특히 아라비아 숫자를 사용하여 수치적인 근삿값을 구하는 알고리즘(Algorithm)을 연구하는 학문이 바로 **수치해석**(Numerical Analysis)이다.

Chapter

03

숫자의 이해와 응용

Understanding and Application of Numbers

학습목표

- 2진수와 10진수의 기본 개념을 이해할 수 있다.
- 부동 소수점 수의 끝수처리와 잘라버리기 개념을 배우고, 차이점을 이해할 수 있다.
- 절대 오차 및 상대 오차의 개념을 이해하고 계산해낼 수 있다.

3장에서는 누구나 일상생활에서 쉽게 사용하는 숫자의 기본 내용을 다룬다. 인간이 데이터를 처리하는 데 사용하는 숫자인 10진수와 컴퓨터가 10진수 데이터를 처리하는 숫자인 2진수의 상관관계를 이해한다.

수치해석의 최종 목표는 추구하는 최종 결과를 아무 오차 없이 해결하는 것이다. 오차를 줄이기 위해서, 즉 결과를 안정적으로 예측하기 위해서는 관련 지식을 공부해야 한다. 디지털 계산, 알고리즘의 정확성과 안정에 대한 연구가 반드시 필요하다.

3.1절에서는 10진 정수와 소수 그리고 2진 정수와 소수에 대한 기본 개념을 배우고, 이들 사이의 상호 관계를 알아본다. 3.2절에서는 10진 부동 소수점 수와 2진 부동 소수점 수의 체계를 분석하고, 부동 소수점 수에 대한 끝수처리와 잘라버리기와 같은 다양한 처리방법을 배운다. 그리고 3.3절에서는 실제값과 근삿값 사이의 절대 오차와 상대 오차가 발생하는 원인을 배우고, 유효숫자의 개념을 학습한다.

3.1 2진수와 10진수의 상관관계

10진법(decimal system)은 아라비아 숫자 0 ~ 9까지 열 개, 10진수(decimal number)를 사용하는 기수법이다. 0보다 크고 1보다 작은 실수인 소수를 표시할 때는 가장 오른쪽에 있는 숫자에 마침표를 사용하고 이후에 다시 0 ~ 9까지의 수를 배치할 수 있다.

2진법(binary notation)은 0과 1 두 개, 2진수(binary number)만을 이용하는 기수법이다. 2진수는 10진수를 2로 나누어서 발생하는 나머지 값을 조합해서 사용한다. 예를 들어 10진수 1을 2로 나눈 나머지는 1이므로 2진수 1이 되고, 10진수 2를 2로 나눈 나머지는 0이므로 2진수 10으로 나타낸다.

10진수와 2진수의 상관관계는 왜 학습해야 하는가? 인간이 손으로 정밀한 수치적 **근삿값**을 계산하기에는 분명 한계가 있다. 이러한 단점을 보완하기 위해서 컴퓨터가 도입되었다. 세계 최초의 전자 컴퓨터는 1946년 미국의 에커트(Eckert)와 모클리(Mauchly)가 미국 육군성의 요청으로 미사일(missile)이 날아가는 각도를 계산하기 위해서 만들었다. 비록 지금 컴퓨터의 성능과 비교할 수 없을 정도로 보잘것없는 수준이었지만 오늘날 인간이 상상하는 것 이상을 보여주는 컴퓨터의 기초가 되었다.

이러한 컴퓨터가 내부에서 데이터를 처리할 때 숫자는 기본적으로 2진수를 이용한다. 따라서 10진수와 2진수의 상관관계는 반드시 이해해야 한다. 10진수가 인간의 숫자라면 2진수는 기계의 숫자 혹은 언어라고 말할 수 있기 때문이다.

3.1.1 2진 정수와 10진 정수

2진 정수(binary integers) B는 다음과 같이 0과 1이 **유한수열** 형태로 표현된다.

$$B = (b_n b_{n-1} \cdots b_2 b_1 b_0)_2 \tag{3.1}$$

식 (3.1)을 다음과 같이 계산하면 10진 정수(decimal integers) D로 변환된다.

$$D = b_n 2^n + b_{n-1} 2^{n-1} + \cdots + b_1 2^1 + b_0 \tag{3.2}$$

예를 들어 2진 정수 $B = (10101)_2$를 10진 정수로 변환하면 다음과 같다.

$$D = 2^4 + 2^2 + 2^0 = 21$$

1이 n개인 2진 정수 $B = (111\cdots1)_2$의 10진 정수는 다음과 같이 간략하게 표현되기도 한다.

$$D = 2^{n-1} + \cdots + 2^1 + 1 = 2^n - 1$$

반대로 10진 정수를 2진 정수로 변환하려면, 우선 식 (3.2)를 2로 나누고 생기는 몫 D_1을 다음과 같이 표현한다. 이때 나머지는 b_0로 표현한다.

$$D_1 = b_n 2^{n-1} + b_{n-1} 2^{n-2} + \cdots + b_1 2^0$$

이를 반복하여 D_1을 2로 나누면, 다시 몫이 생기고 나머지는 b_1이 될 것이다. 이러한 과정을 몫이 0이 될 때까지 반복한다. 매번 나머지로 나오는 0과 1을 역순으로 나열하면 변환된 2진 정수값을 얻을 수 있다.

예를 들어 10진 정수 11을 2진 정수로 변환해보자. 위에서 설명한 대로 계산하면 다음과 같다.

$$\begin{array}{rrrrrr} 2\overline{)11} & = 5 & = D_1 & & & b_0 = 1 \\ & 2\overline{)5} & = 2 & = D_2 & & b_1 = 1 \\ & & 2\overline{)2} & = 1 & = D_3 & b_2 = 0 \\ & & & 2\overline{)1} & = 0 \quad = D_4 & b_3 = 1 \end{array}$$

각 단계마다 구한 나머지를 역순으로 나열하면 $(b_3 b_2 b_1 b_0)_2 = (1011)_2$와 같이 자연스럽게 2진 정수를 얻는다.

MATLAB 3-1 2진 정수와 10진 정수의 상호 변환

매트랩을 이용하여 2진 정수 111을 10진 정수로 변환하라. 또 반대로 10진 정수 7을 2진 정수로 변환하라.

풀이

매트랩에서 2진 정수를 10진 정수로 변환하는 함수는 **bin2dec**이고, 반대로 10진 정수를 2진 정수로 바꾸는 함수는 **dec2bin**이다. 두 함수를 사용하여 실행한 결과는 [그림 3-1]과 같다.

bin2dec 함수를 사용할 때는 반드시 2진 정수를 작은따옴표로 묶어야 한다(❶). 그러나 **dec2bin** 함수를 사용할 때는 작은따옴표를 생략해야만 원하는 값으로 변환된다(❸).

[그림 3-1] 실행 결과

[그림 3-1] ❷번의 괄호 안 '7'은 일반적인 10진수 7이 아닌, 컴퓨터의 아스키 코드(ASCII code) 값으로 지정하고 있는 10진 정수 55를 의미한다. 그러므로 10진 정수 55의 2진 정수값인 110111을 결과값으로 보여준다. 반면에 작은따옴표를 사용하지 않은 ❸번은 그대로 10진 정수 7을 의미하므로, 2진 정수값 111로 변환된다.

3.1.2 2진 소수와 10진 소수

일반적으로 2진 소수(binary fractions) B는 다음과 같이 0과 1이 **무한수열** 형태로 표현된다.

$$B = (.b_1 b_2 b_3 \cdots b_n \cdots)_2 \tag{3.3}$$

식 (3.3)을 10진 소수(decimal fractions) D로 변환시키기 위해 다음과 같이 계산한다.

$$D = b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \cdots \tag{3.4}$$

예를 들어 2진 소수 $B = (.111)_2$를 10진 소수로 변환하면 다음과 같다.

$$D = 2^{-1} + 2^{-2} + 2^{-3} = 0.5 + 0.25 + 0.125 = 0.875$$

반면에 2진 소수 $(.0101010101010)_2$를 10진 소수로 변환하면

$$2^{-2} + 2^{-4} + 2^{-6} + \cdots = 2^{-2}(1 + 2^{-2} + 2^{-4} + \cdots)$$

형태가 된다. 이를 유한 합을 갖는 형태로 표시하려면, 다음과 같은 **등비급수**의 일반 공식을 사용하면 쉽다.

$$\sum_{i=0}^{n} r^i = \frac{1-r^{n+1}}{1-r} \qquad (r \neq 1) \tag{3.5}$$

만일 등비 r의 차수 n이 무한대로 접근하고 $|r|<1$인 조건이라면, 식 (3.5)는 다음과 같이 다시 쓸 수 있다.

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \qquad (|r|<1) \tag{3.6}$$

앞에서 제시한 $2^{-2}(1+2^{-2}+2^{-4}+\cdots)$에 대한 등비는 $|r|=\left|2^{-2}\right|<1$이다. 식 (3.5) 대신 식 (3.6)을 이용하여 계산하면 다음과 같은 10진 소수가 된다.

$$\frac{1}{1-1/4}=\frac{4}{3} \quad \rightarrow \quad 2^{-2}\times\frac{4}{3}=\frac{1}{3}$$

예제 3-1 2진 소수 변환하기

2진 소수 $(.11001100110011\cdots)_2$를 10진 소수로 변환하라.

풀이

주어진 2진 소수는 1100이 계속 반복되는 순환 소수를 나타내고 있다. 순환 소수는 수학적으로 $(0.\overline{1100})_2$로 나타낼 수 있다.

$$(0.\overline{1100})_2 = 2^{-1}+2^{-2}+2^{-5}+2^{-6}+\cdots$$
$$(0.\overline{1100})_2 = 2^{-1}[1+2^{-4}+2^{-8}+\cdots]+2^{-2}[1+2^{-4}+2^{-8}+\cdots]$$
$$(0.\overline{1100})_2 = (2^{-1}+2^{-2})[1+2^{-4}+2^{-8}+\cdots]$$

여기에 식 (3.6)을 이용하면 10진 소수는 0.8이 된다.

$$\frac{1}{1-1/16}\times\frac{3}{4}=\frac{16}{15}\times\frac{3}{4}=\frac{4}{5}=(0.8)_{10}$$

10진 소수를 2진 소수로 변환하기 위해서 식 (3.4)를 다음과 같이 다시 써보자.

$$D = D_1 = b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \cdots$$

여기에 2를 곱하면 b_1 은 정수 부분이 되어 다음 식부터는 사라진다.

$$D_2 = b_2 2^{-1} + b_3 2^{-2} + b_4 2^{-3} + \cdots$$

D_2 에 다시 2를 곱하면 b_2 가 정수 부분이 된다. 이와 같은 과정을 소수 자리가 0이 될 때까지 계속 진행한다. 그리고 차례대로 생성되는 정수 부분을 나열하면 변환된 2진 소수값을 얻을 수 있다.

예를 들어 $D = \frac{1}{5}$ 인 10진 소수를 계산해보자.

$$\begin{aligned}
D &= D_1 = 0.2 \\
2D_1 &= 0.4 = D_2 \rightarrow b_1 = 0 \\
2D_1 &= 0.4 = D_2 \rightarrow b_1 = 0 \\
2D_2 &= 0.8 = D_3 \rightarrow b_2 = 0 \\
2D_3 &= 1.6 = D_4 \rightarrow b_3 = 1 \\
2D_4 &= 1.2 = D_5 = 0.2 = D_1 \rightarrow b_4 = 1
\end{aligned}$$

계속 2를 곱하면 소수 자리가 0이 되지 않고 0011을 반복하는 다음과 같은 2진 순환 소수가 된다.

$$(.001100110011\cdots)_2 = (.\overline{0011})_2$$

매트랩을 이용하여 2진 소수를 10진 소수로 변환하거나 혹은 반대로 변환하는 경우, **Fr_dec2bin** 과 **Fr_bin2dec** 함수를 사용한다. 그러나 이 함수는 고정 소수점 함수가 이미 내장되어 있을 때 사용할 수 있고, 그렇지 않다면 사용할 수 없다. 따라서 매스웍스(MathWorks) 사에서 제공하는 스크립트 파일 Fr_dec2bin.m과 Fr_bin2dec.m을 복사하여 사용 중인 매트랩 디렉터리에 저장해야 사용할 수 있다.

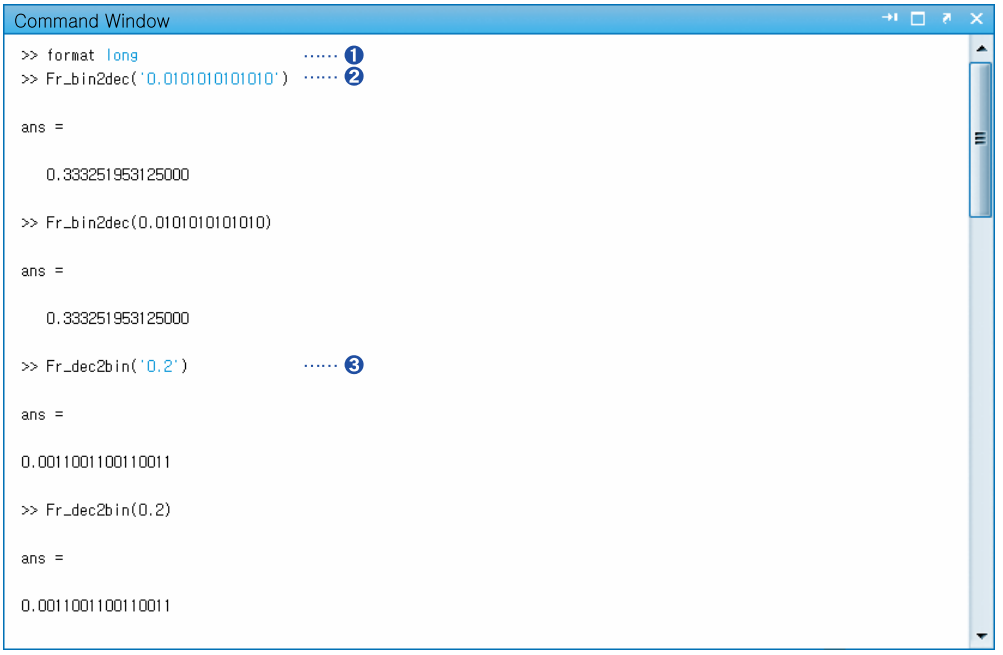
Fr_dec2bin과 **Fr_bin2dec** 함수가 이미 매트랩 작업 공간 디렉터리에 내장된 것으로 가정하고, 이 함수를 사용하여 [MATLAB 3-2]를 실습해보자.

MATLAB 3-2 2진 소수와 10진 소수의 상호 변환

매트랩을 이용하여 2진 소수 $(.0101010101010)_2$을 10진 소수로 변환하라. 그리고 10진 소수 $(0.2)_{10}$를 2진 소수로 변환하라.

풀이

[그림 3-2]에서 2진 소수와 10진 소수의 상호 변환에 사용한 **Fr_bin2dec**(❷)과 **Fr_dec2bin**(❸) 함수는 Fr_bin2dec.m과 Fr_dec2bin.m의 스크립트 파일을 매트랩 디렉터리에 저장한 이후 사용한 결과다. 소수점 자리를 확장하기 위하여 **format long**(❶)을 미리 설정하였다. 2진 정수와 10진 정수와 달리 작은 따옴표를 붙이거나 떼어도 실행 결과는 같다.


Command Window
>> format long ····· ❶
>> Fr_bin2dec('0.0101010101010') ····· ❷
ans =
0.333251953125000
>> Fr_bin2dec(0.0101010101010)
ans =
0.333251953125000
>> Fr_dec2bin('0.2') ····· ❸
ans =
0.0011001100110011
>> Fr_dec2bin(0.2)
ans =
0.0011001100110011

[그림 3-2] 실행 결과

3.2 부동 소수점 수

이 절에서는 컴퓨터가 기본 연산 체계로 사용하고 있는 2진 부동 소수점을 자세히 알아보자. 부동 소수점 체계는 실수를 표현할 때 소수점의 위치를 고정하지 않고, 그 위치를 나타내는 수를 따로 표기하는 방식이다. 컴퓨터에서 사용하면 고정 소수점 체계보다 넓은 범위의 수를 표시할 수 있다.

3.2.1 2진 부동 소수점 수와 10진 부동 소수점 수

컴퓨터 중심의 2진 **부동 소수점 수**(floating-point numbers)를 설명하기 전에 인간에게 더 친근한 10진 부동 소수점 수를 먼저 살펴보자. 0이 아닌 10진수 x를 부동 소수점 수 체계로 일반화하여 부호로 표시하면 다음과 같다.

$$x = \sigma \cdot \bar{x} \cdot 10^{e} \tag{3.7}$$

여기서 σ는 +1 혹은 −1 값을 나타내며, 양과 음을 의미한다. e는 정수로서 소수점의 위치를 나타내는 지수다. 컴퓨터 성능에 따라 차이는 있지만, 지수 e의 범위는 $-99 \le e \le 99$ 범위로 제한되어 있다. $1 \le \bar{x} < 10$의 범위에 있는 $\bar{x}$는 유효숫자 자릿수를 표시하는데, 이를 가수(mantissa)라고 한다. 가수는 부동 소수의 정밀도(precision)를 나타낸다.

예를 들어 분수 $\frac{40}{3}$을 식 (3.7)의 형태로 표시하면

$$\frac{40}{3} = +1 \cdot (1.33333\cdots)_{10} \cdot 10^{1}$$

로 쓸 수 있다. 이때 양의 값을 표시하는 +1은 대개 생략한다.

다른 예로, 1011을 식 (3.7)의 형태로 다시 쓰는 경우를 생각해보자. $\bar{x}$가 1 ~ 9까지 아홉 자릿수를 정수로 사용할 수 있기 때문에 $(1.011)_{10} \cdot 10^{3}$으로 쓸 수 있다. 이제 0이 아닌 2진수 x를 부동 소수점 수 체계로 일반화하여 부호로 표시하면 다음과 같다. 식 (3.7)과 비교하여 가수의 범위는 $\bar{x} = (1)_2$로 표시한다.

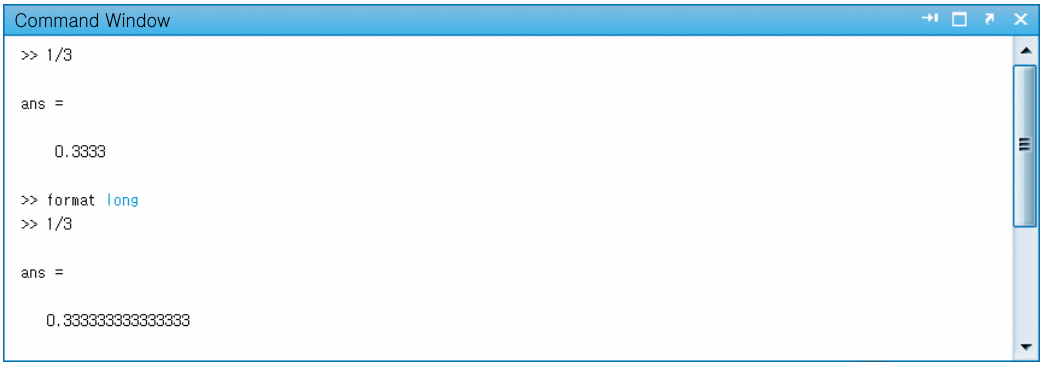
$$x = \sigma \cdot \bar{x} \cdot 2^{e} \tag{3.8}$$

예를 들어 10진 부동 소수점 수 0.1을 식 (3.8)의 형태로 표시해보자. 먼저 10진 부동 소수점 수를 2진

부동 소수점 수로 변환하면 $(0.1)_{10} = (.0001100110011\cdots)_2$ 이 되어 $(1.10011001100\cdots)_2 \cdot 2^{-4}$으로 쓸 수 있다.

분수 $\frac{1}{3}$을 10진 부동 소수점 수로 표시하면, 순환 소수 3이 계속 반복된 형태인 0.3333…이라는 결과가 나온다. 순환 소수가 무한 반복한다는 것은 3을 많이 표시하면 할수록 분수 $\frac{1}{3}$의 실제값에 가까워짐을 의미한다. 공학용 계산기는 성능에 따라 화면에 나타낼 수 있는 자릿수가 한정되어 있다. 만일 화면에 소수 표시가 열 자리까지 가능하다면 0.3333333333으로 보일 것이다. 그러나 계산기의 레지스터(register)에는 식 (3.7)의 형태인 $(3.333333333)_{10} \cdot 10^{-1}$으로 저장될 것이다.

매트랩에서 분수 $\frac{1}{3}$을 입력하여 좀 더 많은 순환 소수 3을 보기 위해서는 [그림 3-3]처럼 **format long**을 지정해야 한다. 그러면 더 많은 3을 표시할 수 있다.



[그림 3-3] 매트랩 이용한 부동 소수점 표시 예

더 많은 소수를 표시하는 것이 실제값에 더욱 근접하여 오차 발생을 줄이는 것과 직결된다. 수치해석을 배우는 가장 큰 이유도 오차를 줄여 **실제값**에 접근한 **근삿값**을 추정하는 것이다. 분수 $\frac{1}{3}$을 식 (3.7) 형태로 다시 써보면 매트랩에서는 $(3.33333333333333)_{10} \cdot 10^{-1}$의 형태로 저장된다.

2진 부동 소수점 수 x가 컴퓨터에 저장되는 형식을 일반화하면

$$FL(x) = \sigma \cdot (1.b_1 b_2 b_3 \cdots b_{23})_2 \cdot 2^e \tag{3.9}$$

와 같이 표현된다. 여기서 $FL(x)$는 2진 부동 소수점 수 x를 컴퓨터 내부에서 기계 숫자로 변형시켜 처리한 부동 소수점의 근삿값을 나타낸다. 음과 양의 부호 표시에 사용하는 σ를 컴퓨터에 저장하는 데 필요한 크기는 1비트다. 가수 표시에 필요한 저장 크기 24비트 중에 2진수 표시에 1비트, 부동 소수점 자리에 23비트가 할당된다. 지수 표시에 필요한 저장 크기는 전체 8비트인데, 이 중 1비트는 음과 양을 표시하고 나머지 정수에 7비트가 사용된다.

2진 부동 소수점 수에서 허용되는 지수의 범위는

$$-(1111111)_2 \le e \le +(1111111)_2$$

이다. 그러나 지수가 0인 2^0을 포함하고 있으므로 실제 범위는 $-126 \le e \le +127$로 표시한다.

기계 입실론(machine epsilon) ε은 컴퓨터로 계산할 때 정밀도를 나타내는 인자로, 1보다 큰 수를 표현하는 데 필요한 소수점 이하의 자릿수다. 예를 들어 y가 컴퓨터 연산에서 1보다 큰 가장 작은 부동 소수라고 하자. y와 ε의 관계를 식으로 나타내면 $\varepsilon = y - 1$이다. 이와 같이 기계 입실론은 컴퓨터 내부에서 표현할 수 있는 모든 수를 정확하게 측정하는 데 널리 사용된다.

식 (3.9)의 형태로 1을 가장 간단한 2진 부동 소수점 수로 표현하면

$$1 = (1.\underbrace{00\cdots0}_{23\text{비트}})_2 \cdot 2^0$$

와 같다. 가수는 전체 24비트를 사용할 수 있는데, 이 조건을 만족하는 기계 입실론은 어떻게 표기할 수 있을까? 부동 소수점 수 아래에서 23비트만 사용할 수 있으므로 1에 2^{-23}을 더한 $\varepsilon = (1.\underbrace{00\cdots01}_{23\text{비트}})_2 \cdot 2^0$이 된다. 그러므로 IEEE 단일 정밀도 부동 소수점 수의 기계 입실론은 다음과 같이 사용한다.

$$\varepsilon = 2^{-23} = 1.19 \times 10^{-7}$$

3.2.2 부동 소수점 수의 끝수처리와 잘라버리기

10진 연산을 먼저 생각해보자. 식 (3.7)을 이용하면 10진 유효숫자가 n자리인 부동 소수점 수 x는 다음과 같이 수정할 수 있다.

$$x = \sigma \cdot (d_1.d_2d_3\cdots d_n)_{10} \cdot 10^e \qquad (3.10)$$

여기서 10진 정수를 나타내는 d_1은 0이 아닌 1 ~ 9 사이의 값으로, $d_2 \sim d_n$까지 10진 소수들은 0을 포함한 0 ~ 9 사이의 값으로 표시된다. 식 (3.10)을 더 일반적인 수의 형태로 표시하면 다음과 같다.

$$x = \sigma \cdot (d_1.d_2d_3\cdots d_nd_{n+1}\cdots)_{10} \cdot 10^e \qquad (3.11)$$

식 (3.11)의 형태는 컴퓨터의 성능과 구조에 따라 사용 가능한 형태로 맞추어져 변환된다. 이러한 맞춤 과정은 **올림 끝수처리**(rounding up)와 **내림 끝수처리**(rounding down) 혹은 **잘라버리기**(chopping) 방법을 이용한다.

식 (3.11)의 x 값에 잘라버리기를 적용하여 컴퓨터에서 허용하는 n자리까지 나타내면 n+1의 자리인 d_{n+1}부터 오른쪽에 자리 잡은 모든 부동 소수 자리를 전부 버린다. 그러므로 잘라버리기를 실

행한 뒤 10진 부동 소수 x가 컴퓨터에 저장되는 형식은

$$FL(x) = \sigma \cdot (d_1.d_2 d_3 \cdots d_n)_{10} \cdot 10^e \tag{3.12}$$

와 같다. 이때 지수는 컴퓨터가 허용하는 영역 내에 맞추어 지정된다.

식 (3.11)의 x 값에 끝수처리를 적용하여 컴퓨터에서 허용하는 n 자리까지만 나타내 보자. $n+1$의 값이 5 ~ 9 사이의 10진수이면 d_n 값에 1을 더한다. 이것을 10진수 올림 끝수처리라고 한다. 반면에 $n+1$의 값이 0 ~ 4 사이의 10진수이면 d_n 값을 그대로 두고, d_{n+1} 부터 오른쪽에 있는 모든 10진 부동 소수점 수는 생략한다. 이것을 10진수 내림 끝수처리라고 한다. 그러므로 끝수처리를 실행한 뒤 10진 부동 소수점 수 x 가 컴퓨터에 저장되는 형식은

$$FL(x) = \begin{cases} \sigma \cdot (d_1.d_2 d_3 \cdots d_n)_{10} \cdot 10^e & (d_{n+1} < 5) \\ \sigma \cdot \left[(d_1.d_2 d_3 \cdots d_n)_{10} + (\underbrace{0.00\cdots1}_{n})_{10} \right] \cdot 10^e & (d_{n+1} \geq 5) \end{cases} \tag{3.13}$$

와 같다. 여기서 $(0.00\cdots1)_{10}$ 항을 식으로 표현하면 10^{-n+1} 이다. 올림, 내림 혹은 잘라버리기를 실행할 때 허용되는 자릿수가 n 이면, n 의 바로 옆 오른쪽 자리인 $n+1$ 값만 비교한다.

이번에는 2진 연산에 대한 끝수처리와 잘라버리기를 살펴보자. 식 (3.8)을 더 일반적인 부동 소수점 수의 형태로 표시하면 다음과 같다.

$$x = \sigma \cdot (1.b_2 b_3 \cdots b_n b_{n+1} \cdots)_2 \cdot 2^e \tag{3.14}$$

여기서 2진 부동 소수점 수의 값은 0 혹은 1 값만을 가진다. 10진 연산과 똑같이 식 (3.14)의 형태도 컴퓨터의 성능과 구조에 의해서 사용 가능하도록 맞추어 변환된다.

식 (3.14)의 x 값에 잘라버리기를 적용하여 컴퓨터에서 허용하는 n 자리까지 나타내면, $n+1$ 의 자리인 b_{n+1} 부터 오른쪽에 자리 잡은 모든 부동 소수점 수 자리를 전부 버린다. 그러므로 잘라버리기를 실행한 뒤 2진 부동 소수점 수 x 가 컴퓨터에 허용되는 형식은

$$FL(x) = \sigma \cdot (1.b_2 b_3 \cdots b_n)_2 \cdot 2^e \tag{3.15}$$

와 같다. 이때 지수는 컴퓨터가 허용하는 영역 내에 맞추어 지정된다.

식 (3.14)의 x 값에 끝수처리를 적용하여 컴퓨터에서 허용하는 n 자리까지 나타낼 때 $n+1$의 값이 1이면 b_n 값에 1을 더한다. 이것을 2진수 올림 끝수처리라고 한다. 반면에 $n+1$의 값이 0이면 b_n 값은 그대로 두고 b_{n+1} 부터 오른쪽에 있는 모든 2진 부동 소수점 수는 생략한다. 이것을 2진수 내림 끝수처리라고 한다. 그러므로 끝수처리를 실행한 뒤 2진 부동 소수점 수 x 가 컴퓨터에 허용되는 형식은

$$FL(x) = \begin{cases} \sigma \cdot (1.b_2 b_3 \cdots b_n)_2 \cdot 2^e & (b_{n+1} = 0) \\ \sigma \cdot \left[(1.b_2 b_3 \cdots b_n)_2 + (\underbrace{0.00\cdots1}_{n})_2 \right] \cdot 2^e & (b_{n+1} = 1) \end{cases} \tag{3.16}$$

와 같다. 여기서 $(0.00\cdots1)_2$ 항을 식으로 표현하면 2^{-n+1} 이다.

10진 정수와 2진 정수의 끝수처리와 잘라버리기도 부동 소수점 수와 같은 방법을 적용하면 된다. 컴퓨터에서 허용하는 정수 자리가 n 이면 정수 자리 $n+1$ 의 10진수 혹은 2진수를 확인하여 실행한다.

예제 3-2 부동 소수점 수 변환하기

10진 부동 소수점 수 $(89.625)_{10}$ 을 손으로 계산하여 2진 부동 소수점 수로 변환하고, 매트랩을 이용해서 확인하라. 그리고 10진 부동 소수점 수 $(0.7)_{10}$ 은 매트랩만을 이용해서 구하라.

풀이

10진 정수 89를 2진 정수로 변환하기 위해 다음과 같이 계산한다.

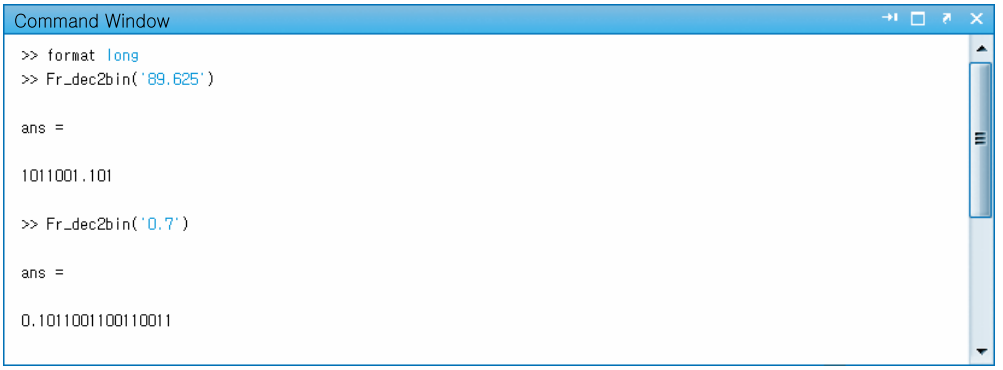
$$\begin{array}{rrrrrrrrl} 2\overline{)89} & = 44 & & & & & & & b_0 = 1 \\ & 2\overline{)44} & = 22 & & & & & & b_1 = 0 \\ & & 2\overline{)22} & = 11 & & & & & b_2 = 0 \\ & & & 2\overline{)11} & = 5 & & & & b_3 = 1 \\ & & & & 2\overline{)5} & = 2 & & & b_4 = 1 \\ & & & & & 2\overline{)2} & = 1 & & b_5 = 0 \\ & & & & & & 2\overline{)1} & = 0 & b_6 = 1 \end{array}$$

2진 정수는 나머지 값을 나타내는 $b_6 \sim b_0$ 까지 차례대로 역순으로 이어서 $(1011001)_2$ 로 변환된다. 10진 부동 소수점 수 0.625를 2진 부동 소수로 변환하려면 다음과 같이 반복한다.

$$\begin{aligned} D &= D_1 = 0.625 \\ 2D_1 &= 1.25 = D_2 \rightarrow b_1 = 1 \\ 2D_2 &= 0.5 \ = D_3 \rightarrow b_2 = 0 \\ 2D_3 &= 1.0 \ = D_4 \rightarrow b_3 = 1 \end{aligned}$$

부동 소수점 수가 0이 되어 더 이상 반복 과정이 필요 없다. 그러므로 2진 부동 소수점 수는 $b_1 \sim b_3$ 까지 차례대로 이어서 $(.101)_2$ 로 변환된다. 변환된 최종값은 2진 정수와 2진 부동 소수점 수를 같이 묶은 $(1011001.101)_2$ 이다.

[MATLAB 3-2]와 같은 방법으로 10진 부동 소수점 수 89.625를 매트랩에서 실행한 결과를 [그림 3-4]에 나타내었다. 0.7을 2진 부동 소수점 수로 변환한 값은 $(.10110011001100\cdots)_2$ 혹은 $(.10\overline{1100})_2$ 이다.



[그림 3-4] 실행 결과

예제 3-3 끝수처리

[예제 3-2]에서 구한 값을 각각 $x=(1011001.101)_2$와 $y=(.101100110011001100\cdots)_2$라고 하자. 5비트 정밀도를 허용하는 컴퓨터에 맞게 끝수처리를 이용하여 x와 y를 근삿값으로 나타내라.

풀이

x와 y를 식 (3.14)의 형태로 다시 써보면 $x=(1.011001101)_2\cdot 2^6$과 $y=(1.011001100\cdots)_2\cdot 2^{-1}$이다. 2진 정수 한 자리와 부동 소수점 수 네 자리를 합친 다섯 자리까지만 나타내려면, 부동 소수점 수 다섯 자리의 값이 0이면 내림을 하고 1이면 올림을 적용하면 된다.

즉 근삿값은 다음과 같다.

$$\hat{x}=(1.0110)_2\cdot 2^6=(1011000)_2$$
$$\hat{y}=(1.0110)_2\cdot 2^{-1}=(.10110)_2$$

예제 3-4 잘라버리기

2비트 2진 컴퓨터를 이용하여 다음을 계산한다고 가정한다.

$$4+2+1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{8}$$

잘라버리기를 이용하여 왼쪽에서 오른쪽으로 더하라.

풀이

2비트 저장 기능만 있는 2진 컴퓨터이므로 식 (3.8) $x=\sigma\cdot\bar{x}\cdot 2^e$의 조건에 맞추어서 문제를 풀어야 한다. 먼저 왼쪽부터 4와 2를 더하면

$$4+2=(10)_2\cdot 2^1+(01)_2\cdot 2^1=(11)_2\cdot 2^1$$

이다. 다시 위의 결과에 1을 더하면 변환된 $1=(01)_2$ 값을 더 이상 지수 2^1으로 표현할 수가 없으므로 잘라버리기를 자동으로 실행한다.

$$4+2+1=(11)_2\cdot 2^1+0$$

같은 과정을 계속 실행하더라도 이미 지정된 지수 2^1으로 더 이상 표시할 수 없으므로 결과는 다음과 같다.

$$4+2+1+\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{8}=(11)_2\cdot 2^1+0+0+0+0+0=6$$

예제 3-5 잘라버리기

[예제 3-4]에 주어진 계산을 잘라버리기를 이용하여 오른쪽에서 왼쪽으로 더하라.

풀이

주어진 식을 오른쪽에서 왼쪽으로 더한다는 것은 다음과 같이 계산함을 의미한다.

$$\frac{1}{8}+\frac{1}{8}+\frac{1}{4}+\frac{1}{2}+1+2+4$$

먼저 왼쪽부터 $\frac{1}{8}$ 과 $\frac{1}{8}$ 을 더하면

$$\frac{1}{8}+\frac{1}{8}=(01)_2\cdot 2^{-3}+(01)_2\cdot 2^{-3}=(01)_2\cdot 2^{-2}$$

이 된다. 이 과정을 반복하면 다음과 같이 계산된다.

$$\frac{1}{8}+\frac{1}{8}+\frac{1}{4}=(01)_2\cdot 2^{-2}+(01)_2\cdot 2^{-2}=(01)_2\cdot 2^{-1}$$

$$\frac{1}{8}+\frac{1}{8}+\frac{1}{4}+\frac{1}{2}=(01)_2\cdot 2^{-1}+(01)_2\cdot 2^{-1}=(01)_2$$

$$\frac{1}{8}+\frac{1}{8}+\frac{1}{4}+\frac{1}{2}+1=(01)_2+(01)_2=(01)_2\cdot 2^1$$

$$\frac{1}{8}+\frac{1}{8}+\frac{1}{4}+\frac{1}{2}+1+2=(01)_2\cdot 2^1+(01)_2\cdot 2^1=(01)_2\cdot 2^2$$

$$\frac{1}{8}+\frac{1}{8}+\frac{1}{4}+\frac{1}{2}+1+2+4=(01)_2\cdot 2^2+(01)_2\cdot 2^2=(10)_2\cdot 2^2=8$$

수학에서는 왼쪽에서 오른쪽으로 더하거나 반대로 오른쪽에서 왼쪽으로 더하더라도 결과는 항상 8이다. 그러나 2비트 2진 컴퓨터라는 조건과 잘라버리기를 실행하면 완전히 다른 결과가 나온다.

예제 3-6 부동 소수점 수 구하기

끝수처리 기능을 가진 컴퓨터의 정밀도를 유효숫자는 $24(n=24)$, 지수의 최소 정수값은 $-126(m=-126)$, 최대 정수값은 $127(M=127)$이라고 가정하자. 이 컴퓨터에 저장되는 0이 아닌 부동 소수점 수는 다음과 같은 정규형으로 나타낼 수 있다.

$$\mathrm{F}:\ \{\text{"0"},\ \pm(1.b_1\ldots b_{23})_2\times 2^e\} \qquad (e\in[-126,\ +127])$$

(a) 가장 큰 부동 소수점 수를 구하라.
(b) 가장 작은 양의 부동 소수점 수를 구하라.
(c) 주어진 정규형 조건을 이용하여 나타낼 수 있는 조합 가능한 모든 부동 소수점 수의 개수를 찾아라.

풀이

(a) 가장 큰 부동 소수점 수는 다음과 같다.

$$(1.\overbrace{1\ldots1}^{23번})_2\times 2^{127}$$

(b) 가장 작은 양의 부동 소수점 수는 다음과 같다.

$$(1.\overbrace{0\ldots0}^{23번})_2\times 2^{-126}$$

(c) 조합 가능한 모든 부동 소수점 수의 개수는 다음과 같이 찾을 수 있다.

$$2^{23}(254)(2)+1$$

여기서 2^{23}은 1.111…1과 1.000…0 사이의 조합 가능한 수를 표시한 것이고, 254는 지수가 −126 ~ 127 사이에 있다는 것을 의미한다. 2는 양과 음 양쪽의 수를 나타내고, 마지막으로 1은 0 값을 표시한다.

3.3 오차

실제값은 x, 컴퓨터로 끝수처리 혹은 잘라버리기를 실행하여 구한 근삿값은 $\hat{x}$라고 하자. 실제값과 근삿값의 차이를 **오차**라고 한다. 오차를 식으로 표현하면 $x-\hat{x}$이다. 이때 x와 $\hat{x}$가 같은 값이면, 오차는 발생하지 않는다($x-\hat{x}=0$).

3.3.1 오차 발생 원인

오차가 발생하는 근본적인 원인은 미지의 현상을 분석하기 위해 수학적으로 모델링하여 계산하는 과정에서 오류가 발생하기 때문이다. 만약 수학적으로 구현된 모델에 오류가 있다면, 결과를 수치적으로 향상시켜서 오차를 줄이기란 쉽지 않다.

컴퓨터가 개발되기 이전 발생했던 오차는 대부분 연산 오류 때문이었으나, 컴퓨터를 사용하기 시작하면서부터는 프로그래밍 오류가 많아졌다. 프로그래밍 오류는 컴퓨터 사용이 범용화된 현재에도 오차가 발생하는 주요 과정이다. 특히 오늘날 프로그래밍 코드는 이전의 프로그래밍보다 더욱 광범위하고 복잡하게 응용되기 때문에, 오류를 찾아내고 수정하는 작업이 매우 어렵다.

실험을 측정하면서 발생하는 오차는 아무리 적은 값이더라도 함부로 제거할 수 없다. 예를 들어, 전자의 반지름은 $(2.81777+\varepsilon)\times10^{-13}\,\mathrm{cm}$인데 전자의 반지름을 이용하여 다른 계산을 실행할 때 오차 ε를 무시하면 정확한 결과가 나오지 않는다. 가능하면 전체 계산의 정확도를 향상시키기 위해서 오차의 영향력을 최소화해야 한다.

3.3.2 절대 오차와 상대 오차

실제값과 근삿값 차이의 절댓값을 **절대 오차**라고 하고, 다음과 같이 쓸 수 있다.

$$\text{절대 오차} = |x-\hat{x}| \tag{3.17}$$

잘라버리기를 적용해서 구한 근삿값은 실제값보다 작으며, 오차는 항상 양의 값을 가진다. 끝수처리 중에 올림을 적용한 경우는 근삿값이 실제값보다 크기 때문에 오차는 항상 음의 값을 가진다. 반면에 내림을 적용한 경우는 항상 양의 값을 가진다.

계산된 오차를 실제값으로 나눈 뒤 절댓값을 적용한 것을 **상대 오차**라고 하고, 다음과 같이 쓸 수 있다.

$$\text{상대 오차} = \left|\frac{x - \hat{x}}{x}\right| \tag{3.18}$$

즉 절대 오차는 오차의 절댓값이고, 상대 오차는 비교 대상에 의해서 크거나 혹은 작게 느껴지는 오차다. 예를 들어 10cm 자로 부엌에 있는 냉장고의 높이를 측정해보니 100cm였다. 냉장고 매뉴얼에 적힌 실제 높이가 105cm라면 절대 오차는 5cm가 된다. 이 절대 오차 5cm는 여의도에 있는 249m 높이의 63빌딩과 비교하면 상대적으로 아주 작은 오차지만, 연필과 비교해보면 상당히 큰 오차가 될 것이다. 결국 절대 오차가 작으면 상대 오차도 작다.

예제 3-7 절대 오차와 상대 오차 구하기

유효숫자 여섯 자리($n = 6$), 지수의 최소 정수값 -40($m = -40$), 최대 정수값 40($M = 40$)과 끝수처리 기능이 있는 컴퓨터가 있다.

(a) 주어진 조건을 고려하여 부동 소수점 수 $2.99792458 \times 10^{10}$(1초당 센티미터로 표시한 빛의 속도)와 $1.67492716 \times 10^{-24}$(그램 단위로 표시한 중성자의 무게)의 근삿값을 구하라.

(b) 절대 오차와 상대 오차를 구하라.

풀이

(a) 구하는 근삿값은 각각 $\hat{x} = 2.99792 \times 10^{10}$와 $\hat{x} = 1.67493 \times 10^{-24}$이다. 유효숫자가 여섯 자리이므로 정수 한 자리와 부동 소수점 수 다섯 자리까지 표시해야 한다. 부동 소수점 수 여섯 자리에서 올림 혹은 내림을 적용하면 첫 번째 경우는 4이므로 내림을 적용하고, 두 번째 경우는 5 이상인 7이므로 올림을 적용한다.

(b) 절대 오차는 식 (3.17), 상대 오차는 식 (3.18)을 이용하면 쉽게 구할 수 있다. $2.99792458 \times 10^{10}$의 절대 오차와 상대 오차는 다음과 같다.

$$|x - \hat{x}| = |2.99792458 \times 10^{10} - 2.99792 \times 10^{10}| = 0.00000458 \times 10^{10} = 4.58 \times 10^{4}$$

$$\left|\frac{x - \hat{x}}{x}\right| = \left|\frac{4.58 \times 10^{4}}{3.00 \times 10^{10}}\right| = 1.52666666 \times 10^{-6} = 1.53 \times 10^{-6}$$

상대 오차를 계산할 때는 먼저 구한 절대 오차의 유효숫자와 같은 자리로 표시해야 한다.
$1.67492716 \times 10^{-24}$의 결과는 다음과 같다.

$$|x - \hat{x}| = |1.67492716 \times 10^{-24} - 1.67493 \times 10^{-24}| = 0.00000284 \times 10^{-24} = 2.84 \times 10^{-30}$$

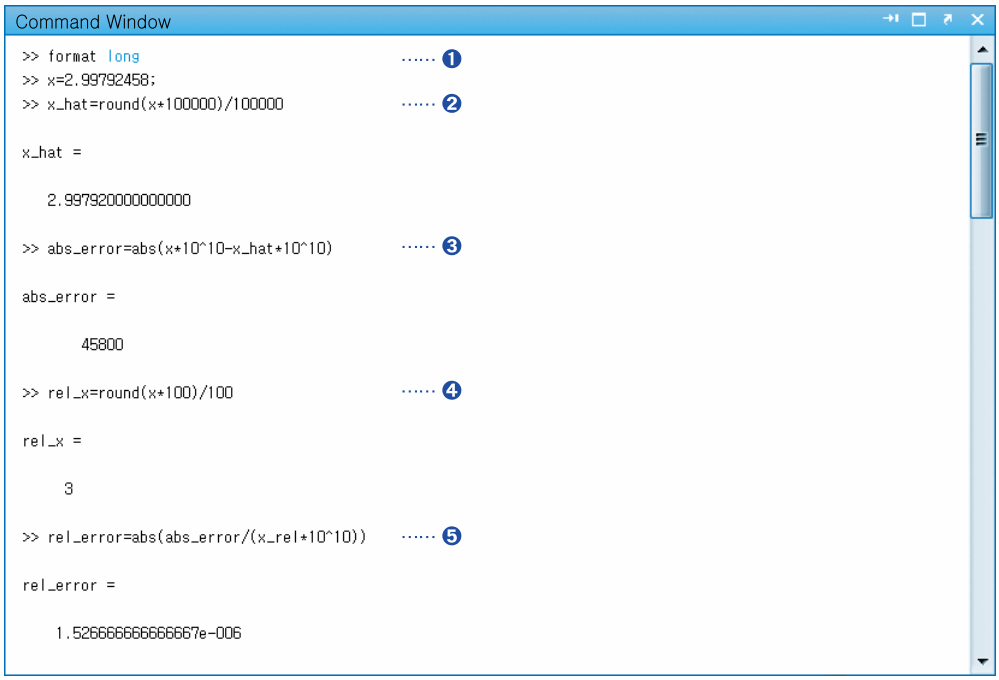
$$\left|\frac{x - \hat{x}}{x}\right| = \left|\frac{2.84 \times 10^{-30}}{1.67 \times 10^{-24}}\right| = 1.7005988 \times 10^{-6} = 1.70 \times 10^{-6}$$

MATLAB 3-3 round 함수로 절대 오차와 상대 오차 구하기

매트랩을 이용하여 [예제 3-7]에서 제시한 $2.99792458 \times 10^{10}$(1초당 센티미터로 표시한 빛의 속도)에 대한 절대 오차와 상대 오차를 구하라.

풀이

매트랩에서 **round** 함수를 이용하면 올림 혹은 내림을 실행할 수 있으나 사용이 제한적이다. 지수승을 제외한 부동 소수점 수에 대한 끝수처리를 **round** 함수로 먼저 실행한 이후에 지수승을 다시 곱해서 표현할 수 있다. 매트랩 실행 결과는 [그림 3-5]와 같다.



[그림 3-5] 실행 결과

매트랩의 디폴트(default) 출력 형식은 부동 소수점 수 네 자리까지만 표시하므로 주어진 실제값을 나타내기 위해서는 **format long**(❶)의 형식으로 변환해야 한다. 출력 형식 **format long**은 열다섯 자리까지 표시된다. 실제값에 대한 끝수처리를 실행할 때 부동 소수점 수 다섯 자리까지 표시하므로 100000을 곱하고 다시 100000을 나누어 주는 방법을 이용하였다(❷).

근삿값과 실제값에 편의상 제거한 10의 10승값을 다시 곱하고 실제값에서 근삿값을 뺀 후에 절댓값을 실행하면 절대 오차를 구할 수 있다(❸). 식 (3.10)에 맞추어서 절대 오차를 다시 표현하면 4.58×10^{4}이 되고, 상대 오차에 사용하는 실제값을 다시 절대 오차의 유효숫자 세 자리에 맞추면 3.00×10^{10}이 된다(❹).

절대 오차를 3.00×10^{10}으로 나누고 절댓값을 실행하면 상대 오차를 구할 수 있다(❺). 이때 계산된 상대 오차도 3자리 유효숫자로 다시 맞추어 조정하면 손으로 계산한 1.53×10^{-6}의 값을 얻는다.

예제 3-8 부동 소수점의 근삿값 구하기

다음 물음에 답하라.

(a) [예제 3−2]에서 10진 부동 소수점 수 $x = (89.625)_{10}$ 과 $y = (.7)_{10}$ 에 대한 2진 부동 소수점 수를 각각 $x = (1011001.101)_2$ 와 $y = (.101100110\ldots)_2$ 로 변환하였다. 5비트 정밀도와 끝수처리 기능이 있는 컴퓨터를 이용하여 변환된 x 와 y의 2진 부동 소수점의 근삿값을 구하라.

(b) 10진 정수 x 와 y의 절대 오차와 상대 오차를 유효숫자가 두 자리인 부동 소수점 수로 나타내라.

풀이

(a) $x = (1011001.101)_2$의 근삿값은 2진 정수 여섯 자리에서 내림이 적용되어 $\hat{x} = (1011000)_2 = (88)_{10}$ 가 되고, $y = (.101100110\ldots)_2$ 의 근삿값은 2진 부동 소수점 수 여섯 자리에서 내림이 적용되어 $\hat{y} = (.10110)_2 = (.6875)_{10}$ 가 된다.

(b) 절대 오차는 식 (3.17), 상대 오차는 식 (3.18)을 이용한다. x의 절대 오차와 상대 오차는 다음과 같다.

$$|x - \hat{x}| = |89.625 - 88| = 1.625$$

$$\left|\frac{x - \hat{x}}{x}\right| = \left|\frac{1.625}{89.625}\right| = 0.0181311 = 1.8 \times 10^{-2}$$

그리고 y 의 결과는 다음과 같다.

$$|y - \hat{y}| = |0.7 - 0.6875| = 0.0125$$

$$\left|\frac{y - \hat{y}}{y}\right| = \left|\frac{0.0125}{0.7}\right| = 0.01785714 = 1.8 \times 10^{-2}$$

3.3.3 유효숫자 오차의 손실

실제값을 추정하기 위해 근삿값을 계산할 때, **유효숫자**(significant figure)는 대단히 중요하다. 일반적으로 유효숫자는 0이 아닌 숫자를 말한다. 예를 들어 22와 22.3의 유효숫자는 각각 두 자리와 세 자리다. 컴퓨터나 계산기를 이용하여 식을 계산할 때, 유효숫자에 따라 오차가 작게 혹은 크게 발생할 수 있다.

입력변수 x 에 대한 결과 함수 $f(x)$를 다음과 같이 나타낼 수 있다고 가정하자.

$$f(x) = x[\sqrt{(x+1)} - \sqrt{x}]$$

변수 x 의 값을 1부터 10배씩 증가시켜 100000까지 입력한다. (여기서 사용하려는 계산기는 여섯 자리의 10진수만 표시할 수 있다.)

연산의 우선 순위에 맞추어 $x=1$ 일 때를 계산해보자. 함수 $f(x)$를 구하는 연산 과정은 다음과 같다.

❶ x에 1을 더하고 제곱근을 구한다.
❷ x에 대한 제곱근을 구한다.
❸ 과정 ❶에서 구한 제곱근에서 과정 ❷에서 구한 제곱근을 뺀다.
❹ 과정 ❸의 계산 결과에 x의 값을 곱한다.

실제 함수값을 계산하기 위해서는 과정마다 끝수처리를 적용하지 않고, 모든 과정이 끝난(❹) 이후에 끝수처리를 한다. 여섯 자릿수를 맞추면 다음과 같은 계산 결과를 얻는다.

$$f(1)=1\cdot[\sqrt{(2)}-\sqrt{1}]=[1.4142135623-1]=0.4142135623=0.414214$$

반면에 근사시킨 함수값을 계산하기 위해서는 과정마다 끝수처리를 적용한다. 즉 다음과 같은 계산 결과를 얻는다.

$$f(1)=1\cdot[\sqrt{(2)}-\sqrt{1}]=[1.41421-1]=0.414210$$

같은 방법으로 입력값을 10배씩 증가시켜서 계산하면 [표 3-1]과 같은 결과를 얻는다. 입력값이 증가하면 할수록 근삿값과 실제값 사이에 엄청난 오차가 발생하는 것을 볼 수 있다.

만일 유효숫자를 여섯 자리에서 열 자리로 증가시키고 x의 입력값 100000을 입력하면 근삿값은 158.1200000이 되고, 실제값은 158.1134878이 되어 유효숫자를 여섯 자리만 사용한 [표 3-1]의 결과와 다르게 작은 오차를 유지한다. 유효숫자를 증가시키면 오차의 손실을 최소화할 수 있다.

[표 3-1] 오차 손실 보기 : 유효숫자를 증가시킨 결과

x	근삿값 $f(x)$	실제값 $f(x)$
1	0.414210	0.414214
10	1.54340	1.54347
100	4.99000	4.98756
1000	15.8000	15.8074
10000	50.0000	49.9988
100000	100.000	158.113

예제 3-9 유효숫자의 기초 개념

다음에 주어진 10진수의 유효숫자를 구하라.

0.046, 7.90, 8200

풀이

부동 소수점 수인 경우 0이 아닌 수들의 앞에 있는 0은 유효숫자에 포함하지 않는다. 따라서 0.046에서는 4와 6만이 유효숫자가 된다. 반면에 0이 아닌 수들의 뒤에 있는 부동 소수점 0은 유효숫자로 처리한다. 따라서 7.90의 유효숫자는 세 자리다.

10진 정수 8200의 경우는 8.200×10^3, 8.20×10^3, 8.2×10^3와 같이 세 자리 형태로 표현할 수 있다. 각각의 경우 유효숫자는 혹은 네 자리, 세 자리, 두 자리라고 말할 수 있다. 이러한 불확실성을 방지하기 위해서는 식 (3.10)과 같이 표시하는 것이 바람직하다.

계산기에는 유효숫자의 허용 범위가 낮아서 발생하는 오차를 줄이기 위해 **정밀도**가 설정되어 있다. 이를 조정하는 것은 쉽지 않다. 지정된 유효숫자의 변환 없이 삼각함수의 유사성을 이용하여 오차를 줄이는 방법은 다음과 같다.

입력변수 x에 대한 결과 함수 $f(x)$를 다음과 같이 나타낼 수 있다고 가정하자.

$$f(x)=\frac{1-\cos x}{x^2} \qquad (x\neq 0)$$

변수 x의 호도값(radian value)을 0.1부터 10배씩 감소시켜서 0.00001까지 입력한다. 이때 주어진 함수의 연산을 위해서 사용하려는 계산기는 유효숫자를 열 자리만 표시할 수 있다.

연산의 우선 순위에 맞추어서 $x=0.01$일 때를 계산해보자. 함수 $f(x)$를 구하는 연산 과정은 다음과 같다.

❶ cos(0.01)의 값을 구한다.
❷ 1에서 과정 ❶의 결과를 뺀다.
❸ (0.01)의 제곱값을 구한다.
❹ 과정 ❷를 과정 ❸으로 나눈다.

실제 함수값을 계산하기 위해서는 과정마다 끝수처리를 적용하는 것이 아니고, 모든 과정이 끝난 (❹) 이후에 끝수처리를 한다. 10자리를 맞추면 다음과 같은 계산 결과를 얻는다.

$$f(0.01)=\frac{1-\cos(0.01)}{(0.01)^2}=\frac{1-0.9999500004166652}{0.0001}=0.4999958333$$

반면에 근사시킨 함수값을 계산하기 위해서는 과정마다 끝수처리를 적용한다. 즉 다음과 같은 계산 결과를 얻는다.

$$f(0.01)=\frac{1-\cos(0.01)}{(0.01)^2}=\frac{1-0.9999500004}{0.0001}=0.4999960000$$

같은 방법으로 입력값을 10배씩 감소시켜서 계산하면 [표 3-2]와 같은 결과를 얻는다. 0.1 ~ 0.0001까지는 근삿값과 실제값의 오차가 줄어들지만, 0.00001의 값을 입력하면 갑자기 오차가 커진다. 이런 경우 문제점을 찾는 것도 힘들지만, 문제 해결도 쉽지 않다.

[표 3-2] 오차 손실 보기 : 입력값을 감소시킨 결과

x	근삿값 $f(x)$	실제값 $f(x)$
0.1	0.4995834700	0.4995834722
0.01	0.4999960000	0.4999958333
0.001	0.5000000000	0.4999999583
0.0001	0.5000000000	0.4999999996
0.00001	0.0	0.5000000000

주어진 함수 $f(x)=\frac{1-\cos x}{x^2}$ 는 오차를 줄이기가 쉽지 않다. 그러나 다음과 같이 삼각함수의 유사성을 이용하면 오차를 쉽게 줄일 수 있다.

$$\cos(2\theta)=2\cos^2(\theta)-1=1-2\sin^2(\theta)$$

위의 식을 $x=2\theta$ 로 치환하여 원래 식을 다음과 같이 수정할 수 있다.

$$f(x)=\frac{1-\cos x}{x^2}=\frac{2\sin^2(x/2)}{x^2}$$
$$=\frac{1}{2}\left[\frac{\sin(x/2)}{x/2}\right]^2$$

수정된 식을 이용하여 입력값 0.01에 대한 근삿값을 다시 계산하면 다음과 같이 실제값과 오차가 줄어든다.

$$f(0.01)=\frac{1}{2}\left[\frac{\sin(0.01/2)}{0.01/2}\right]^2=\frac{1}{2}\left[\frac{0.004999979167}{0.005}\right]=\frac{1}{2}[0.9999916668]$$
$$=0.4999958334$$

예제 3-10 유리화를 이용한 유효숫자의 처리

방정식 $ax^2+bx+c=0$ 의 근을 구하기 위해 보편적으로 사용하는 공식은 다음과 같다.

$$x_{\pm}=\frac{-b\pm\sqrt{b^2-4ac}}{2a} \qquad (3.19)$$

식 (3.19)의 분자와 분모에 $-b \mp \sqrt{b^2 - 4ac}$ 를 곱하면 다음과 같은 공식을 얻는다.

$$x_{\pm} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \tag{3.20}$$

3자리 유효숫자로 끝수처리 연산을 실행하는 컴퓨터로 식 (3.19)와 (3.20)을 각각 사용하여 이차방정식 $x^2 - 15x + 1 = 0$ 의 근을 계산하라.

풀이

식 (3.19)를 이용하면

$$x_{\pm} = \frac{-(-15) \pm \sqrt{(-15)^2 - 4(1)(1)}}{2(1)} = \frac{15 \pm 14.9}{2}$$

$x_+ = 15.0$ 과 $x_- = 0.0500$ 이다. 식 (3.20)을 이용하면

$$x_{\pm} = \frac{-2(1)}{(-15) \pm \sqrt{(-15)^2 - 4(1)(1)}} = \frac{-2}{-15 \pm 14.9}$$

$x_+ = 20.0$ 과 $x_- = 0.0669$ 이다.

실제값은 $x_+ = 14.9333$ 과 $x_- = 0.0670$ 이다. 실제값과 비교하여 식 (3.20)을 이용한 경우가 오차가 더 작다. 식 (3.20)은 분자의 유리화라고 하는데, 이 공식은 유효숫자의 크기에 의해서 발생하는 오차를 줄이는 데 이용한다.

3.3.4 언더플로우 오류와 오버플로우 오류

식 (3.7)과 식 (3.8)에 표시된 지수는 상한값과 하한값을 가진다. 이는 컴퓨터의 처리 장치가 취급할 수 있는 수치 연산 범위를 지정하고 있다는 의미다. 즉 하나의 수치를 표현하기 위해서 컴퓨터에 할당된 기억 용량은 일정 크기로 정해져 있다.

부동 소수점 수가 너무 커서 컴퓨터에서 허용하는 지수의 상한값을 넘겨서 생기는 오류를 **오버플로우 오류**(overflow error)라고 한다. 반대로 부동 소수점 수가 너무 작아 컴퓨터에서 허용하는 범위가 되지 못할 때 생기는 오류를 **언더플로우 오류**(underflow error)라고 한다. 이는 처리 장치가 취급하는 절대치의 하한보다도 더욱 작은 연산 결과로 마이너스 지수가 생긴 경우다.

예를 들면 [예제 3-7]을 지수의 최소 정수값이 $-20(m = -20)$이고, 최대 정수값이 $5(M = 9)$인 컴퓨터로 다시 문제를 푼다고 가정해보자. 빛의 속도에 대한 근삿값의 지수승은 10^5보다 커 오버플로우 오류가 발생하고, 중성자에 대한 근삿값의 지수승은 10^{-20}보다 작아 언더플로우 오류가 발생할 것이다.

Chapter 03 핵심요약

01. 수치해석의 목적

수치해석을 학습하는 주요 목적은 수치적으로 측정된 근삿값을 좀 더 실제값과 일치시키는 알고리즘을 연구하는 것이다.

02. 컴퓨터의 언어 : 2진수

측정한 10진수 데이터를 수치해석을 위해 컴퓨터에 입력하면, 컴퓨터 내부에서는 2진수 데이터인 기계 숫자로 변환되어 사용된다.

03. 2진수를 10진수로 변환

- 2진 정수 $(b_n b_{n-1} \cdots b_2 b_1 b_0)_2$ 를 10진 정수로 변환하기

$$D = b_n 2^n + b_{n-1} 2^{n-1} + \cdots + b_1 2^1 + b$$

- 2진 소수 $(.b_1 b_2 b_3 \cdots b_n \cdots)_2$ 를 10진 소수로 변환하기

$$D = b_1 2^{-1} + b_2 2^{-2} + b_3 2^{-3} + \cdots$$

04. 부동 소수점 수 표시

- 일반화된 10진수 x 의 부동 소수점 수 표시

$$x = \sigma \cdot \overline{x} \cdot 10^e$$

σ	양과 음의 전체 부호 표시
e	소수점의 위치를 표시하는 정수 지수
$\overline{x}$	유효숫자 자릿수

- 일반화된 2진수 x 의 부동 소수점 수 표시

$$x = \sigma \cdot \overline{x} \cdot 2^e$$

σ	양과 음의 전체 부호 표시
e	소수점의 위치를 표시하는 정수 지수
$\overline{x}$	유효숫자 자릿수

05. 부동 소수점 수

• 일반화된 10진 부동 소수점 수

$$x = \sigma \cdot (d_1.d_2 d_3 \cdots d_n d_{n+1} \cdots)_{10} \cdot 10^e$$

잘라버리기	d_{n+1} 이하 소수점 무조건 버리기
내림 끝수처리	$d_{n+1} < 5$, d_{n+1} 이하 소수점 버리기
올림 끝수처리	$d_{n+1} \geq 5$, d_n 자리에 1 더하기

• 일반화된 2진 부동 소수점 수

$$x = \sigma \cdot (1.b_2 b_3 \cdots b_n b_{n+1} \cdots)_2 \cdot 2^e$$

잘라버리기	b_{n+1} 이하 소수점 무조건 버리기
내림 끝수처리	$b_{n+1} = 0$, b_{n+1} 이하 소수점 버리기
올림 끝수처리	$b_{n+1} = 1$, b_n 자리에 1 더하기

06. 절대 오차와 상대 오차

• 절대 오차는 실제값과 근삿값의 차이를 말한다.

$$\text{절대 오차} = |x - \hat{x}|$$

• 상대 오차는 절대 오차를 실제값 $|x|$ 로 다시 나눈 값을 말한다.

$$\text{상대 오차} = \left|\frac{x - \hat{x}}{x}\right|$$

07. 유효숫자

0이 아닌 숫자로써 지정되는 개수를 말하며, 이에 따라 근삿값과 실제값 사이의 오차 폭이 작아지거나 혹은 커진다.

Chapter 03 연습문제

3.1 10진 부동 소수점 수 $(97.3125)_{10}$ 을 손으로 계산하여 2진 부동 소수점 수로 변환하고 매트랩을 이용해서 확인하라. 10진 부동 소수점 수 $(.1625)_{10}$ 은 매트랩을 이용해서 변환된 값을 찾아라.

3.2 문제 3.1에서 구한 값을 $x = (1100001.0101)_2$ 라고 하자. 5비트 정밀도를 허용하는 컴퓨터에 맞게 x를 끝수처리하여 근삿값으로 표시하라. 그리고 10진수를 사용하여 절대 오차와 상대 오차를 계산하라.

3.3 2진 부동 소수점 수 $(11.1101)_2$ 와 $(0.10001)_2$ 를 10진 부동 소수점 수로 변환하라. 매트랩을 이용하여 결과를 확인하라.

3.4 숫자의 2진수 표시는 $x = \sigma \cdot \bar{x} \cdot 2^e$ 와 같이 나타낸다. $11_2 + 0.1_2 + 0.1_2$ 의 덧셈 계산을 잘라버리기와 끝수처리로 계산하라. 필요한 모든 계산 과정을 보이고 소수점 두 자리까지 표시하라.

3.5 $0.1_2 + 0.1_2 + 11_2$ 의 덧셈 계산을 잘라버리기와 끝수처리로 계산하라. 필요한 모든 계산 과정을 보이고 소수점 두 자리까지 표시하라(문제 3.4의 덧셈을 반대로 실행하는 문제).

3.6 다음 계산을 2비트 2진 컴퓨터로 계산한다고 가정한다.

$$8+4+2+1+\frac{1}{2}+\frac{1}{4}+\frac{1}{4}$$

(a) 잘라버리기를 이용하여 왼쪽에서 오른쪽 방향으로 덧셈을 실행하라.
(b) 잘라버리기를 이용하여 오른쪽에서 왼쪽 방향으로 덧셈을 실행하라.

3.7 다음 계산을 2비트 2진 컴퓨터로 계산한다고 가정한다.

$$\overbrace{1+1+\cdots+1+1}^{100번}$$

(a) 잘라버리기를 이용하여 왼쪽에서 오른쪽 방향으로 덧셈을 실행하라.
(b) 잘라버리기를 이용하여 오른쪽에서 왼쪽 방향으로 덧셈을 실행하라.

3.8 IEEE 2배 정밀도 표준은 유효숫자 53, 지수의 최소 정수값은 −1022이고, 최대 정수값은 1023이다. 이런 구조의 컴퓨터에 저장되는 0이 아닌 부동 소수점 수는 다음과 같은 정규형으로 나타낼 수 있다.

$$F : \{"0", \pm(1.b_1 \ldots b_{52})_2 \times 2^e\} \qquad (e \in [-1022, +1023])$$

(a) 가장 큰 부동 소수점 수를 구하라.

(b) 가장 작은 양의 부동 소수점 수를 구하라.

(c) 주어진 정규형을 이용하여 조합 가능한 모든 부동 소수점 수들의 개수를 찾아라.

3.9 유효숫자 5자리, 지수의 최소 정수값 −25, 최대 정수값 25와 끝수처리 기능이 있는 컴퓨터가 있다.

(a) 부동 소수점 수 $x = 1.673534 \times 10^{-19}$ 와 $y = 6.0221367 \times 10^{26}$ 의 근삿값을 주어진 조건에 맞추어서 계산하라.

(b) 절대 오차와 상대 오차를 구하라.

3.10 매트랩을 이용하여 문제 3.9의 x 에 대한 절대 오차와 상대 오차를 구하라.

3.11 이차방정식 $ax^2 + bx + c = 0$의 근이 $a \neq 0$이면 다음 공식들을 이용하여 구할 수 있다.

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}, \qquad x_2 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

이차방정식 $x^2 - 14x + 1 = 0$의 정확한 근을 계산하면 각각 $x_1 = 13.9282$와 $x_2 = 0.0718$이다. 이때 유효숫자가 세 자리인 끝수처리 연산을 실행하는 컴퓨터로 근삿값을 찾아라. 그리고 각각의 상대 오차를 계산하라.

3.12 문제 3.11에서 보여준 근들의 공식을 분자에 대한 유리화 형태로 변형하면 다음과 같다.

$$x_1 = \frac{-2c}{b + \sqrt{b^2 - 4ac}}, \qquad x_2 = \frac{-2c}{b - \sqrt{b^2 - 4ac}}$$

유효숫자가 세 자리인 끝수처리 연산을 실행하는 컴퓨터로 이차방정식 $x^2 - 14x + 1 = 0$에 대한 근들의 근삿값을 계산하라. 각각의 상대 오차를 계산하고 문제 3.11에서 구한 근들의 근삿값과 비교하라.