第2章　数の体系，コンピュータ，浮動小数点数

伊藤　　「たった 20 坪だけど」
設計士　「20.14 坪ですよ」
伊藤　　「4000 万もしたけど」
不動産屋「4321.5 万円ですよ」
伊藤　　「家 3000 万もかかるけど」
工務店員「2880 万です」

伊藤理佐「やっちまったよ一戸建て」(双葉社)

本章では，数値計算で扱う「数 (number)」とはどのようなものかを，コンピュータというものの氏素性と共に確認することにする。数学における数は，無限集合として取り扱われるものであるが，有限量のメモリしか持ち得ないコンピュータに無限個の数を扱うことはできず，そこにはどうしても理論との齟齬が生じる。そこを如何にうまくゴマカすか，ということを本章では解説し，ゴマカした結果生じた齟齬によって如何なる問題が生じるか，ということについては次章で解説する。

2.1　数の体系

小学，中学，高校で学んできた数学において使用してきた「数 (number)」は 5 種類あった。自然数 (natural number)，整数 (integer)，有理数 (rational number)，実数 (real number)，複素数 (complex number) である。これらの数は全て無限個数存在し，それぞれ集合 (set) としてカテゴライズされ，それぞれ $\mathbb{N}$(自然数の集合), $\mathbb{Z}$(整数の集合), $\mathbb{Q}$(有理数の集合), $\mathbb{R}$(実数の集合), $\mathbb{C}$(複素数の集合) と太字の大文字で表記される。この 5 種類の数はどのような性質を持っていたか，簡単に復習しておくことにする。

自然数の集合 $\mathbb{N}$ を本書では

$$\mathbb{N} = \{0, 1, 2, ..., n, ...\}$$

と定義する。0 は自然数に含めないことが多いが，含んでいても困ることはない。また，自然数は他の集合の要素の添え字 (index) としてよく用いられるが，その際には 0 も使うことが増えてきており，利便性を高める意味でも自然数の一員としてカウントしておくことにする。

$\mathbb{N}$ は加算 (+)，乗算 ($\times$ 又は $\cdot$ 又は省略) に関して閉じている，という性質を持っている。即ち，

$$\left.\begin{array}{rcl} a+b & \in & \mathbb{N} \\ a \cdot b & \in & \mathbb{N} \end{array}\right\} \text{for } \forall a, b \in \mathbb{N}$$

となる。つまり，任意の自然数同士の加算及び乗算の結果は必ず自然数となる。

整数の集合 $\mathbb{Z}$ は

$$\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\} = \{..., -2, -1\} \cup \mathbb{N}$$

と定義される。自然数の負 (minus) の整数を追加した集合となっており，加算，乗算だけでなく，減算 (–) に関しても閉じている。

有理数の集合 $\mathbb{Q}$ は，これ以上約分できない既約分数 (irreducible fraction) の集合である。分子 (numerator) として $\mathbb{Z}$，分母 (denominator) として 0 を除いた自然数の集合 $\mathbb{N}^* = \mathbb{N} - \{0\}$ を取ると，これらの直積 $\mathbb{Q} = \mathbb{Z} \times \mathbb{N}^*$ として定義されたものと見ることもできる (表 2.1)。

表 2.1: 有理数の集合

分子 → / 分母 ↓	⋯	−2	−1	0	1	2	⋯
1	⋯	−2	−1	0	1	2	⋯
2	⋯	−1	−1/2	0	1/2	1	⋯
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

$\mathbb{Q}$ は加算，乗算，減算だけでなく，割る数として 0 を除いた除算 (/) に関しても閉じている。

既約分数は 10 進小数 (decimal fraction) としても表現できるが，有限桁に収まる小数，即ち有限小数 (terminating decimal fraction) になるものと，同じ桁パターンが繰り返し無限に連なる循環 (無限) 小数 (recurring decimal fraction) になるものとに分かれる。

例えば，有限小数 0.3145 は

$$\frac{3145}{10000} = \frac{629}{2000}$$

という既約分数の表現であるが，循環小数 $0.314531453145\cdots$ は

$$\begin{aligned} 0.314531453145\cdots &= \lim_{n\to\infty} \sum_{i=1}^{n} 3145 \cdot 10^{-4i} \\ &= 3145 \lim_{n\to\infty} \sum_{i=1}^{n} 10^{-4i} \\ &= 3145 \cdot \frac{10^{-4}}{1 - 10^{-4}} \\ &= \frac{3145}{9999} \end{aligned}$$

という既約分数の表現である。いずれの小数も $\mathbb{Q}$ の要素である。

では，循環しない無限小数，例えば

$$\begin{aligned} \sqrt{2} &= 1.41421356\cdots \\ \pi &= 3.1415923653\cdots \\ e &= 2.718281\cdots \end{aligned}$$

はどうなるのか。これらはいずれも既約分数を四則演算を用いて無限個組み合わせることによって得られるものである (計算方法は第 6 章を参照のこと)。しかしこれらはもはや既約分数として表現できず，$\mathbb{Q}$ の要素ではない。従って，$\mathbb{Q}$ は無限回の四則演算に関しては閉じているとは言えない。このような極限値 (limit value) を考えるためには循環しない無限小数，つまり無理数 (irrational number) も含む数の集合，即ち実数が不可欠となったのである。

実数の集合 $\mathbb{R}$ は，$\mathbb{Q}$ と無理数の集合を合わせたもので，幾何学的には数直線 (real line) と同一視することが多い (図 2.1)。

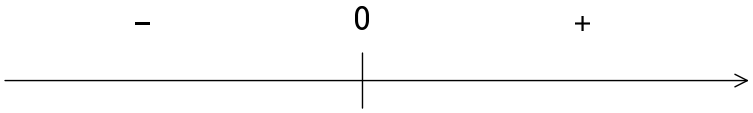


図 2.1: $\mathbb{R}$ の表現としての数直線

$\mathbb{R}$ を構成することによって，四則演算及び極限操作全てに関して閉じた数の集合が完成したことになる。微分積分は実数の存在なくしては成り立たない理論体系なのである。しかしこれにもまだ不備がある。

実数を係数とする代数方程式 (第 13) 章参照) の解は必ずしも実数の範囲に収まらない。よく知られているように，2 次 (代数) 方程式

$$x^2 - 3x + 4 = 0$$

の解は $\sqrt{-7}$，即ち，2 乗すると -7 という負の実数になる要素を含まなくてはならない。即ち，実数係数 (実係数) の n 次代数方程式 ($n \in \mathbb{N} < \infty$) の解に関しては，$\mathbb{R}$ は閉じていないことになる。閉じるためには新たな数を考案する必要が出てくる。

そこで，$\sqrt{-7} = \sqrt{7}\cdot\sqrt{-1}$ と実数とそうでない成分に分離し，$\sqrt{-1}$ を新たに虚数単位 (imaginary unit)[1] と命名することにして，実数とは異なる数，即ち虚数 (imaginary number) を作ることにする。こうして複素数 $\mathbb{C}$ は $\mathbb{R}$ の直積として

$$\mathbb{C} = \left\{ a + b\sqrt{-1} \;\middle|\; a, b \in \mathbb{R} \right\}$$

のように，実数と実係数の虚数との和として表現される。幾何学的には $\mathbb{R}^2$ 平面と同一視されることが多く，これを Gauss 平面と呼ぶ (図 2.2)。

複素数の係数 (複素係数) を持つ n 次代数方程式の解は，重複も込めて必ず n 個存在し，全て $\mathbb{C}$ の要素となる (代数方程式の基本定理)。よって，$\mathbb{C}$ は四則演算，極限操作，代数方程式の解の全てに関して閉じていることになる。

こうして，$\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$ という包含関係を持つ 5 種類の数の性質を示してきたが，これらは全て無限個の要素から成る無限集合である。このうち，$\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ は，$\mathbb{N}$ の要素と一対一対応 (単射) が存在する，即ち自然数の添え字を付けることが可能な可算 (countable) 無限集合であるのに対し，$\mathbb{R}$ と $\mathbb{C}$ は非可算 (uncountable) 無限の集合である。

[1] 虚数単位はアルファベット小文字の i もしくは j で代用することが多いが，i も j も他の用途に使用されることが多く紛らわしいので本書では使用しない。

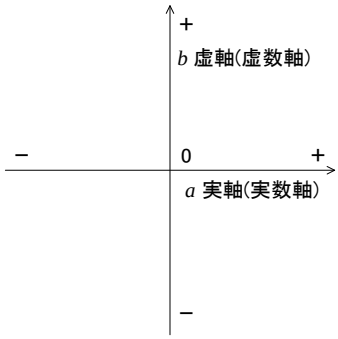


図 2.2: $\mathbb{C}$ の表現としての $\mathbb{R}^2$ 平面

また，$\mathbb{Q}$ のうち循環小数として表現されるものや無理数は有限小数として表現することは出来ない。これらの性質はキレイな理論体系の中で扱う分には便利だが，実際に現在のコンピュータで扱おうとすると途端に厄介な問題を引き起こすことになる。

2.2　コンピュータの構成, bit, byte, 2 進表記の自然数

現在の，ごく一般的なパーソナルコンピュータ (Personal Computer, PC) の模式図を図 2.3 に示す。

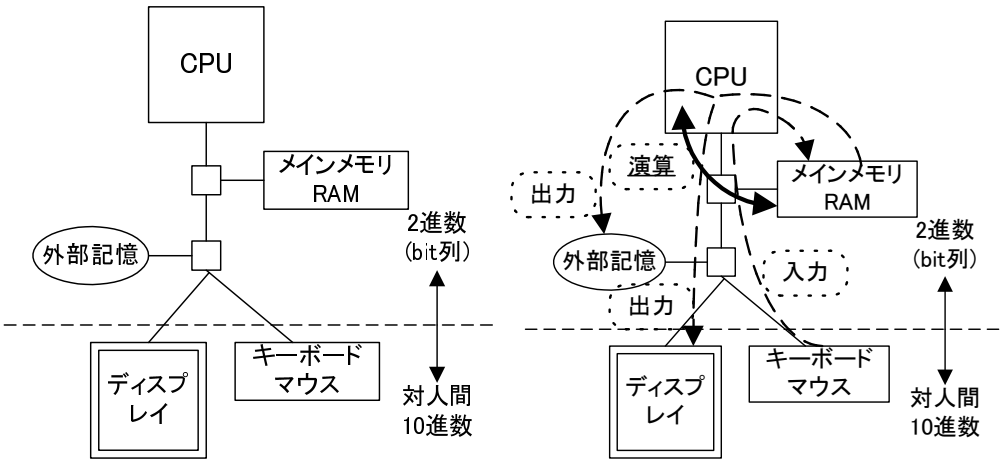


図 2.3: PC の構成 (左) と処理の流れ (右)

コンピュータ内部における処理の流れを簡単に述べると次のようになる。

外部記憶 (HDD，光ディスク等) に保存されているファイル (データの塊) や，キーボード・マウスといった入力装置からデータが送り込まれ，一度メインメモリ (RAM) に格納される。処理の途中あるいは最後には，処理されたデータが外部記憶やディスプレイへ出力される。この内部にお

けるデータのやり取りは，プリント基板に焼き付けられた銅線，バス (bus) を介して行われ，データは数百 MHz(10^6Hz)〜数 GHz 単位のデジタル電気信号 (clock) に乗って届けられる。ここで言うデジタル (digital) とは，自然数の 0 か 1 のどちらかに相当する 2 値信号で，バス一本につき一桁分が 1clock 毎に届けられる。従って，この一本のバスに乗る一桁分の 0 もしくは 1 という値が，コンピュータにおけるデータの最小単位ということになり，これを 1bit(ビット, b) と呼ぶ。このバスは最低 8 本分，つまり 8b あり，これを 1byte(バイト, B) と呼ぶ。現在の PC はこのバスが 32b(4B)〜64b(8B) 分あるのが普通である。よって，コンピュータで扱うデータは，このバスに乗る，0 または 1 という 2 値に変換可能な電気信号となるものでなければならない。つまり，1B のデータは 1clock で 8b の 0, 1 の塊，例えば

$$01011010$$

と表現できるようなものでなければならない。これは自然数を 2 進数 (binary number) として表現したものと同一視できる。上のデータは

$$0 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 90$$

から，$90 \in \mathbb{N}$ と見ることが出来る。従って，1B で表現可能なデータの範囲は

2 進表現		10 進表現
00000000	$\Longleftrightarrow$	0
00000001	$\Longleftrightarrow$	1
$\vdots$		$\vdots$
11111110	$\Longleftrightarrow$	254
11111111	$\Longleftrightarrow$	$255 = 2^8 - 1$

となる。この 256 個の自然数に，アルファベット・数字・記号を対応させたのが ASCII コードである。このように，文字，画像，色情報 ••• コンピュータで扱う全てのデータは，bit 数の長短の違いはあれ，自然数を 2 進表記したものと同一視される bit 列として表現されるものになっており，入出力の際には 10 進 ↔2 進の変換が伴うのである。

2.3 固定小数点数と浮動小数点数

コンピュータの中では全てのデータが，0 もしくは 1 の 2 値を一桁とする bit 列になっている。では，実数はどのようにして bit 列として表現すればよいのだろうか？

前述したように，$\mathbb{R}$ は非可算無限集合である。それに対し，コンピュータが扱うことが出来るのは bit 数によって制限される有限の自然数の集合までである。当然，有限可算集合の要素と，非可算無限集合の要素との間に一対一対応を作ることは不可能であり，よって，コンピュータが実数を正確に表現することも不可能ということになる。

理論的に不可能なことをやろうとすれば，そこには必ず妥協が入ることになる。しかし，どうせ妥協するならば，工学的な利便性を伴うように妥協する方法を考えることで，多少の理論的齟齬は目をつぶってもらえるのではないだろうか。その結果生まれたのが浮動小数点数 (floating-point

number) という，近似 (approximation) 方式である。コンピュータが誕生して以来，半世紀を経た今でも実数を bit 列で表現する方法として不動の位置を占めている。

実数を bit 列にする方法は，2 段階の理論的齟齬を経由する。

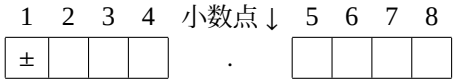
まず，使用できる bit 数に応じて，表現できる実数の範囲を有限の範囲に制限する。もし計算結果がこの範囲を超える実数になった場合は，桁あふれ，オーバーフロー (overflow) として無限大 (±∞, ±Inf) の扱いをする。無限大になる値を含んだ計算結果は数値でない値，即ち非数 (Not-a-Number, NaN) として，特別な bit パターンが割り当てられる。

次に，使用できる bit 数に応じて桁数を決め，全ての実数をこの桁数の小数に落とし込む，つまり近似するのである。「近い」値になるように「似せ」てはいるが，元の実数とは違うものになることが多い。この操作を丸め (round-off) と呼び，この近似によって生じる元の値 (真値) とのずれを丸め誤差 (round-off error) と呼ぶ。この誤差ついては次章で詳しく述べることにする。

この，実数を有限桁の小数に近似する方法としては，固定小数点 (fixed-point) 方式と浮動小数点方式があるが，前述のしたように現在はもっぱら後者が用いられる。これらの近似によって得られる実数をそれぞれ固定小数点数 (fixed-point number)，浮動小数点数 (floating-point number) と呼ぶ。

コンピュータ内部におけるデータ表現は bit 列であるが，説明を簡単にするために以降は 10 進表現を用いてこの 2 つの近似方式について解説する。基数 (base) が 2 であろうと 10 であろうと，その本質は変わらないからである。

例えば，10 進 8 桁しか使えない状況を考えよう。固定小数点方式の場合は，先頭に ± の符号を表わす桁を確保し，残りの桁を小数として扱う。小数点は中央付近に固定 (fixed) しておく。例えば，符号桁を除いて上位桁から 3 桁目に小数点を配置すると次のようになる。

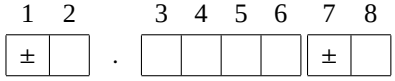


この場合，表現可能な実数の範囲は

$$
\begin{aligned}
&-999.9999\\
&\quad\vdots\\
&-0.0001\\
&\qquad 0\\
&+0.0001\\
&\quad\vdots\\
&+999.9999
\end{aligned}
$$

となる。±1000 以上の実数は全て overflow して処理され，無限大と同じ扱いとなる。これはいかにも範囲が狭い。

それに対し，浮動小数点方式は，小数部分 (仮数部, mantissa) の他に指数部 (exponent) を設け，基数の指数部ベキ乗を掛け合わせて小数点を浮遊 (floating) させるようにする。例えば，先と同じ 10 進 8 桁しか使えない場合で，符号部に 1 桁，仮数部に 5 桁，指数部に 2 桁 (1 桁分は指数の符号部) 割り当てたとする。小数点は仮数部の一桁目と二桁目の間に置くと，次のような形になる。



この場合，表現可能な実数の範囲は

$$
\begin{aligned}
&-9.9999 \cdot 10^{+9} \\
&\quad\vdots \\
&-1.0000 \cdot 10^{-9} \\
&-0.9999 \cdot 10^{-9} \\
&\quad\vdots \\
&-0.0001 \cdot 10^{-9} \\
&\qquad 0 \cdot 10^{0} \\
&+0.0001 \cdot 10^{-9} \\
&\quad\vdots \\
&+0.9999 \cdot 10^{-9} \\
&+1.0000 \cdot 10^{-9} \\
&\quad\vdots \\
&+9.9999 \cdot 10^{+9}
\end{aligned}
$$

となり，同じ桁数でも，固定小数点数方式に比べて圧倒的に広い範囲の実数を扱うことが出来る (図 2.4)。指数部の桁数を増やせば更にその差が広がるが，仮数部の桁数が減るため近似の精度 (precision) は落ちる。浮動小数点数の桁数とは，この仮数部に割り当てられた桁数を指す。今の場合は 10 進 5 桁の浮動小数点数，ということになる。

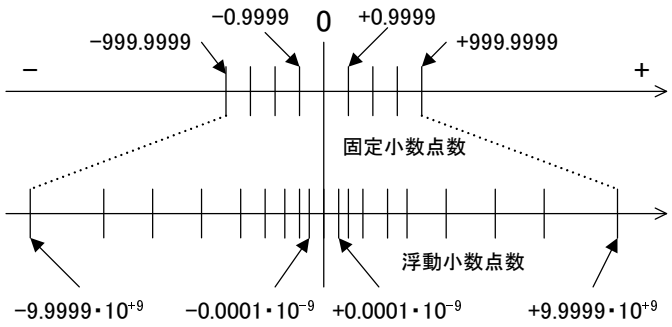


図 2.4: 固定小数点数と浮動小数点数の表現可能範囲

以降は，この浮動小数点方式による実数の近似方法，浮動小数点数のみ扱うことにする。現在広く使用されている浮動小数点数は 2 を基数とする IEEE754 規格 (IEEE754 stadard)(次章参照) である。しかし，人間が直接扱うには不都合であるため，入出力にはもっぱら 10 進表記が用いられる。本書でも説明のための例題は 10 進表記を採用するが，実際に計算されるのは 2 進に変換されたものであることを常に忘れてはならない。

最後に，この 10 進 5 桁の浮動小数点数を用いた計算を実行してみよう。

$$\frac{1}{3} + \frac{9}{2}$$

もちろんこの答えは 29/6 である。この真値を覚えておこう。

計算は，加算される 2 数を浮動小数点数に変換してから実施される。まず 1/3 は

$$\begin{aligned}\frac{1}{3} &= 0.33333\cdots \\ &\downarrow \text{正規化 (仮数部の桁を合わせる)} \\ &= 3.33333\cdots\times 10^{-1} \\ &\downarrow \text{丸め (仮数部 6 桁目を四捨五入)} \\ &\approx 3.3333\cdot 10^{-1}\end{aligned}$$

と変換される。10 進 5 桁に収まらない場合は，このように仮数部 6 桁目以降を丸める。具体的には，10 進数の場合，6 桁目が 5 以上であれば 5 桁目に 1 を加えて 6 桁目以降を切り捨て，4 以下であればそのまま切り捨てる。これを四捨五入方式と呼ぶ。この方式は丸め誤差を極力小さくすることができるため，RN(Round to Nearest) 方式とも呼ばれる。

9/2 も同様に

$$\begin{aligned}\frac{9}{2} &= 4.5 \\ &= 4.5000\cdot 10^{0}\end{aligned}$$

と変換される。この場合は丸め誤差は発生しない。

次に加算を行う。指数部を大きい方に揃えてから実行する。

$$\begin{array}{rcl} 3.3333\cdot 10^{-1} & = & 0.33333\cdot 10^{0} \\ & +) & 4.5000\ \ \cdot 10^{0} \\ \hline & & 4.8333\boxed{3}\cdot 10^{0} \\ & & \downarrow \text{丸め} \\ & & 4.8333\boxed{\ }\cdot 10^{0} \end{array}$$

こうして，$4.8333\cdot 10^{0}$ という結果を得る。

以上をまとめると，浮動小数点数同士の演算は

1. 2 数を入力し，浮動小数点数に変換する
2. 演算して，桁数に収まらなければ丸める
3. 演算結果の出力

となる。実際には更に，入力時には 10 進 →2 進変換，出力時には 2 進 →10 進変換が行われ，演算も 2 進演算で実行される。

このように，コンピュータにおける数値計算は，外部から与えるデータと，実際に計算されるデータの形式が異なるものであり，入力時，あるいは演算実行時には必ず変換及び丸めという操作が実行され，**近似値による近似計算が行われている**ということになる。従って，真値による正しい実数計算とのずれが生じることになるが，有限桁の 2 進浮動小数点数のみを扱うようにしたことで，非常に高速な計算が可能になっているという「工学的利便性」がもたらされたことも見逃してはならない。近似という理論的齟齬と，演算の高速性という工学的利便性，この二者は互いに trade-off(あちらが立てばこちらが立たず) の関係にあるのである。

2.4 丸め方式

任意の実数を有限桁の浮動小数点数丸める方法としては，10 進数の場合，四捨五入方式と切り捨て方式がある。例えば 10 進 5 桁の浮動小数点数を使用している時に，±1.23456 という値が与えられると，四捨五入方式では ±1.2346 へ，切り捨て捨て方式では ±1.2345 へ丸められる。この場合，どちらの方式でも符号によらずに丸められた後の浮動小数点数が決定される。

この 2 つの方式を任意の p 進数 ($p > 0$ は偶数) の場合で言うと，$p/2 - 1$ 捨 $p/2$ 入方式と切り捨て方式ということになる。

$p/2 - 1$ 捨 $p/2$ 入方式は，丸める前の実数 a と距離的に最も近い浮動小数点数に丸める (Round to Nearest) 方式である。よってこれを RN 方式と呼び，常に RN 方式で丸められる浮動小数点演算の状態を RN モード (RN mode) と呼ぶ。

それに対して，切り捨て方式は数直線 (図 2.5) で言うと，常に 0 に近い方向へ丸める (Round to Zero)，という方式であると言える。従って，これを RZ 方式と呼び，この方式で常に丸められる状態を RZ モード (RZ mode) と呼ぶ。

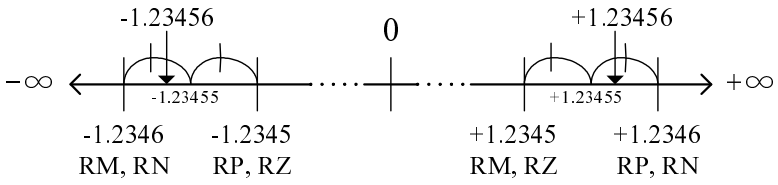


図 2.5: 丸め方式による違い

これ以外にも，現在の浮動小数点演算の主流である IEEE754 規格 (IEEE754 standard) においてはこれ以外に二つの丸め方式 (RM, RP) が取り入れられており，合わせて 4 つの丸めモード (RN, RZ, RM, RP) の設定ができるようになっている。実際の IEEE754 規格は 2 進法ベースであり，RN 方式については偶数丸め (次章参照) という方式をとっていて，厳密な 0 捨 1 入計算ではないが，実用的にはそれに近いものと考えてよい。これをまとめたものを図 2.5 に示す。

追加された RM, RP モードは丸められる実数の符号によって変化する方式である。これを例で示そう。

RM 方式 (モード) は常に $-\infty$ 方向へ丸める (Round to Minus infinity) 方式である。この場合は −1.23456 は −1.2346 へ，+1.23456 は +1.2345 へ丸められる。

RP 方式は，常に $+\infty$ 方向へ丸める (Round to Plus infinity) 方式である。この場合は逆に −1.23456 は −1.2345 へ，+1.23456 は +1.2346(RN 方式と同じ) へ丸められる。

このように，RM, RP 方式では実数の符号によって丸めの方向が変化する。

この二つの丸めモードが追加された理由は，精度保証を行う時に使用される厳密な区間演算 (Interval arithmetic) に不可欠な機能だったからである。しかし，区間演算のみで計算を進めていくと，実際の誤差より過大になり過ぎる傾向が見られるため，現在の精度保証計算では，アルゴリズムごとに工夫して単純な区間演算は少なくすることが望ましいとされている。

なおこの 4 つの丸めモードを全て使用すると，厳密な区間演算にはならないが，かなり近い丸め誤差の測定が可能である [17]。数値例は第 16 章を参照されたい。

演習問題

1. $a = 2.3456 \times 10^2$, $b = 3.1415$ である時，$a + b$, $a - b$, ab, a/b の値を 10 進 5 桁の浮動小数点演算によって求めよ。なお，丸めは四捨五入方式で行うものとする。

2. 一次方程式 $\sqrt{2}x - 3 = 2/3$ を，10 進 5 桁の浮動小数点演算で求めよ。

3. 32b で表現可能な自然数の範囲を求めよ。また 64b で表現可能な自然数の範囲も求めよ。

4. 任意の循環小数 $0.a_1a_2a_3a_4a_1a_2a_3a_4\cdots = 0.\dot{a_1}\dot{a_2}\dot{a_3}\dot{a_4}$ を分数形に変化する方法について説明し，実際に循環小数 $0.\dot{1}\dot{2}\dot{3}\dot{4}$ を既約分数形に変換せよ。

5. 1 坪 (ツボ) は，一辺の長さが 1 間 (ケン) の正方形の面積のことである。1 間は 6 尺 (シャク) であり，1 尺は 10/33 メートル (m) である。20.14 坪は何平方メートル (m^2) か，10 進 5 桁の浮動小数点数として答えよ。

6. $\pm\sqrt{2}$ を RN, RZ, RM, RP 方式でそれぞれ 10 進 5 桁の浮動小数点数に丸めよ。