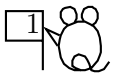
数値計算講義　第1 回
オーバーフロー・ アンダーフロー・ 桁落ち・ 情報落ち

金子　晃

kanenko@mbk.nifty.com
http://kanenko.a.la9.jp/

計算機は分数と小数のどちらを得意とするか?

計算機は有理数と実数のどちらを得意とするか?



cf. 昔の電卓では小数の計算はできても分数の計算はできなかった:
$1 \div 3 = 0.33333333$ となってしまった.
分数が扱える電卓は 1980 年代始めに現れた.
ヨーロッパに持っていったら, 使い捨て懐炉と並びびっくりされた.
(ちなみに, 君達の世代は小学校で分数電卓を使っていたらしく,
電卓に対する感覚が我々とは全然違うようです. (^^;)

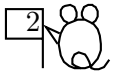
小数と実数の違いを御存じですか?

問題1 1 と $0.99999\cdots$ は同じもの? それとも違うもの?

ところで, 分数と有理数の違いを御存じですか?

問題2 $\dfrac{1}{2}$ と $\dfrac{2}{4}$ は同じもの? それとも違うもの?

問題2 に対する答



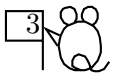
分数とは $\frac{q}{p}$, $p, q \in \boldsymbol{Z}$, $p \neq 0$ の形の記号の全体.
これは (q, p) と書いても差し支えない. i.e. サイズ2 の整数配列.
だから, 計算機で**誤差無しに, 正確に**取り扱える (ただしオーバーフローに注意)

有理数とは, 分数の同値類 $(q, p) \sim (s, r) \Longleftrightarrow ps = qr$
だから, $\frac{1}{2}$ と $\frac{2}{4}$ は分数としては異なるものだが, 同一の有理数を表す.

小学校では, $\frac{2}{4}$ は $\frac{1}{2}$ にしないと減点される.
高等学校くらいになると, それだけではあまり減点されることは無くなる.
$\Longrightarrow$ 有理数としての値が合っていれば良いという立場に変わる.

【 参考】 計算機で普通に扱える整数の範囲
16 ビットで表現できる数 ($65536 = 2^{16}$):
☆ 符号付き2 バイト整数 (int) $-32768 \sim 32767$
☆ 符号無し 2 バイト整数 (unsigned int) $0 \sim 65535$
32 ビットで表現できる数 ($4294967296 = 2^{32}$):
☆ 符号付き 4 バイト整数 (long) $-2147483648 \sim 2147483647$
☆ 符号無し 4 バイト整数 (unsigned long) $0 \sim 4294967295$
これを越えると整数のオーバーフローとなる.

計算機の中は二進法



基本用語

1 ビット = 二進法の1 桁のこと

例: 十進法の 8 は二進法で 1000
$19 = 16 + 2 + 1$ は二進法で 10011
$100 = 64 + 32 + 4$ は二進法で 1100100

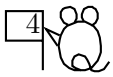
このように二進法では長く なり 過ぎるので十六進法を 併用する
十六進法は数字が十六種類必要:

```
0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
```

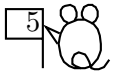
1 バイト = 十六進2 桁 = 二進8 桁 = 8 ビット
(計算機科学におけるデータの基本単位)
例: 二進 $_{\text{MSB}}\underbrace{0110}_{4\text{桁}}\ \underbrace{1101}_{4\text{桁}}{}_{\text{LSB}}$ = 十六進 6D = 十進 $6 \times 16 + 13 = 109$

十進	二進	十六進
0	0	0
1	1	1
2	10	2
3	11	3
4	100	4
5	101	5
6	110	6
7	111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F
16	10000	10

十進	二進	十六進
17	10001	11
18	10010	12
19	10011	13
⋮	⋮	⋮
31	11111	1F
32	100000	20
33	100001	21
⋮	⋮	⋮
63	111111	2F
64	1000000	30
65	1000001	31
⋮	⋮	⋮
127	1111111	7F
128	10000000	80
⋮	⋮	⋮
255	11111111	FF
256	100000000	100



問題1 に対する答



小数とは数字を並べたもの： $b_m b_{m-1}\cdots b_1 b_0.a_1 a_2\cdots$

例： 十進小数の場合, 数字は $0,1,2,3,4,5,6,7,8,9$ を使う.

$\left.\begin{array}{l} 123.4567 \quad \text{(有限小数)} \\ 0.123123123\cdots \quad \text{(循環小数)} \end{array}\right\} \cdots$ 有理数

$3.1415926535\cdots$ (循環しない無限小数) $\cdots$ 無理数

有限小数は $123.4567 = 123.456700000\cdots$ と考えると循環小数の一種

実数とは四則と連続性を表現した公理系により定義されるもので,
小数はその一つの表現である.

$$\text{十進小数} \quad 0.a_1a_2\cdots \quad \text{は} \quad \frac{a_1}{10}+\frac{a_2}{10^2}+\cdots \quad \text{の意}$$

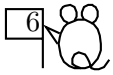
普通は対応は一対一だが, 有限小数になる実数に限り 2 種の表現を持つ：

$0.2 = 0.200000\cdots = 0.199999\cdots$ 後者は $\dfrac{2}{10} = \dfrac{1}{10}+\dfrac{9}{10^2}+\dfrac{9}{10^3}+\cdots$ の意味で,
無限等比級数を加えると, 結局有理数 $\dfrac{1}{5}$ に帰する.

二進法で表すと $\dfrac{1}{5}$ は一意な循環小数となる：

$$\frac{1}{5} = 0.00110011\cdots = \frac{1}{2^3}+\frac{1}{2^4}+\frac{1}{2^7}+\frac{1}{2^8}+\cdots$$

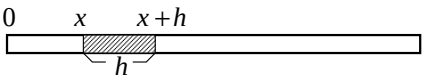
問題 (i) 十進法で循環小数となる数は, 二進法でも循環するか？
(ii) 十進法で有限小数となる数は, 二進法でも有限小数となるか？



★ $\lim_{h\to 0}\dfrac{f(x+h)-f(x)}{h}$ は $h\to 0$ のとき $f'(x)$ に収束するか？

物理実験をしてみよう！
線密度 ρ の鉄の棒が有る．
線密度とは，単位長さ当たりの質量のこと

$$\rho(x)=\frac{m(x+h)-m(x)}{h}$$



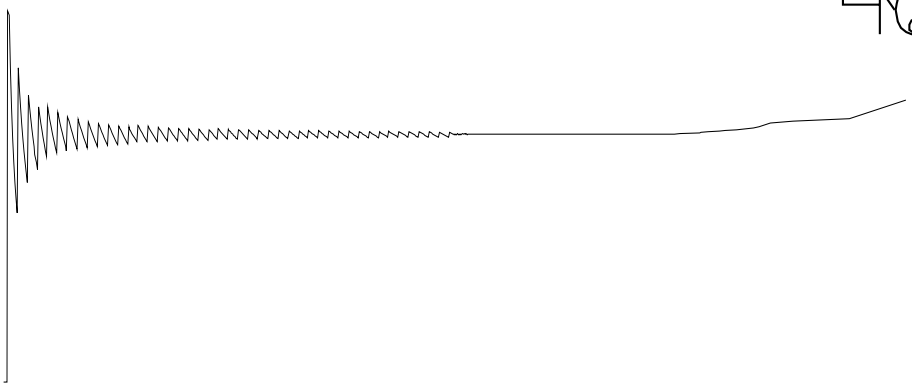
右図の部分の平均線密度

一様に作られていないので，密度は場所により異なる．
ここで，できるだけ短く切って薄片の重さを精密に測り，
点 x における“ 真の ”線密度を測定する．(これが微分の定義！)

わが大学には猛烈に精密なカッターが有り，
理想的な実験が可能である．(^^;

Q 以前に文系の学生を相手に総合科目でこういう話をしたら，
“ そのカッター普段はどんな実験に使うんですか ”という質問が
出ましたけど，もちろん冗談ですよ〜．(^^;;;;;;

実験結果のグラフ（^^;）

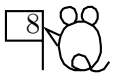


h を小さくして行くと, 確かに一定の値に近付いて行く.

もっと小さくすると次第に振動を始める.
あまり薄く切り過ぎたため, 棒の分子を通り過ぎるときに
分子数が不連続的に変化するためだろう.

最後はめちゃめちゃになっちゃった!

物理学と数学のリテラシーの差 (おまけ)(^^;)



物理では, 微分するときに h をあまり小さくしてはいけない.
分子が見えるようになる直前で止める!
そこから先は, h を 0 に近付けてはいけない. (統計物理の世界になっちゃう.)
現実から離れて, ここまでの実験結果を $h \to 0$ に外挿すると,
数学で学んだ微分積分学が使えるようになる.

例 流体の基礎方程式 **Navier-Stokes 方程式** (河村研より拝借 (^^;))
流体の運動は多数の粒子の複雑な運動である.
それを, 適当な h のサイズで見ると, 次のような微分方程式で記述できる.

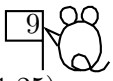
$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} + \frac{1}{\rho} \nabla p - \nu \Delta \boldsymbol{u} = \boldsymbol{f}$$

計算機による数値計算の約半分は, この方程式の近似解を求める仕事である.

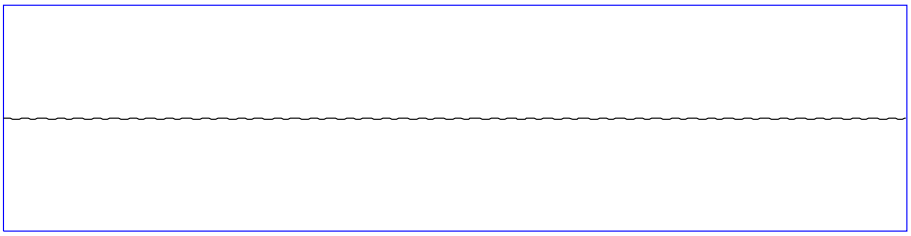
銀河は多数の恒星の集まりである.
遠くから見ると渦運動をしている.
$\Longrightarrow$ 流体の運動論が応用できる.

微分とは, $h \to 0$ としたときの
$\dfrac{f(x+h) - f(x)}{h}$ の極限である.
このときの h の適切なサイズはどれくらいだろう ?!

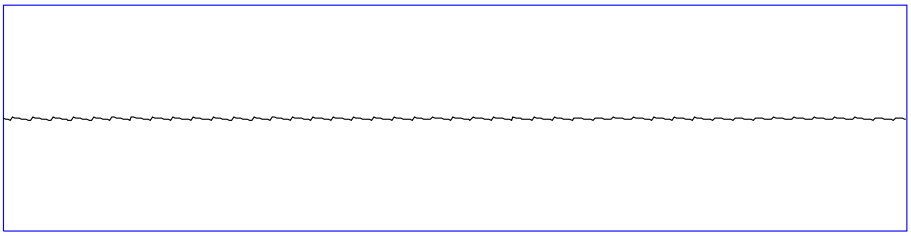
$f(x) = x$ の導関数計算の倍精度浮動小数による実行結果



$h \to 0$ のとき $\dfrac{(1+h)-1}{h}$ を計算してみる. (縦軸の範囲は $0.75 \sim 1.25$)

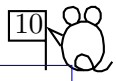


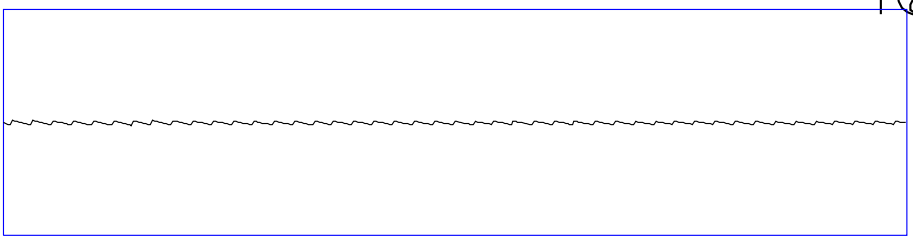
$10^{-7} \leq h \leq 10^{-7} + 10^{-14}$ (200 等分で描画)
相当大きなところでも, 描画のための整数化の際の切り 捨て処理のせいで
1 ピクセルくらいの揺れは生じる.



$3 \times 10^{-14} \leq h \leq 4 \times 10^{-14}$ (200 等分で描画)
何だかちょっと心配なグラフになってきた.

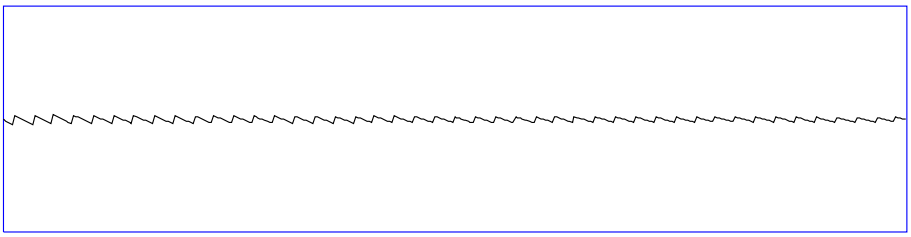
うーん, なんだかなー





$2 \times 10^{-14} \leq h \leq 3 \times 10^{-14}$ (200 等分で描画)

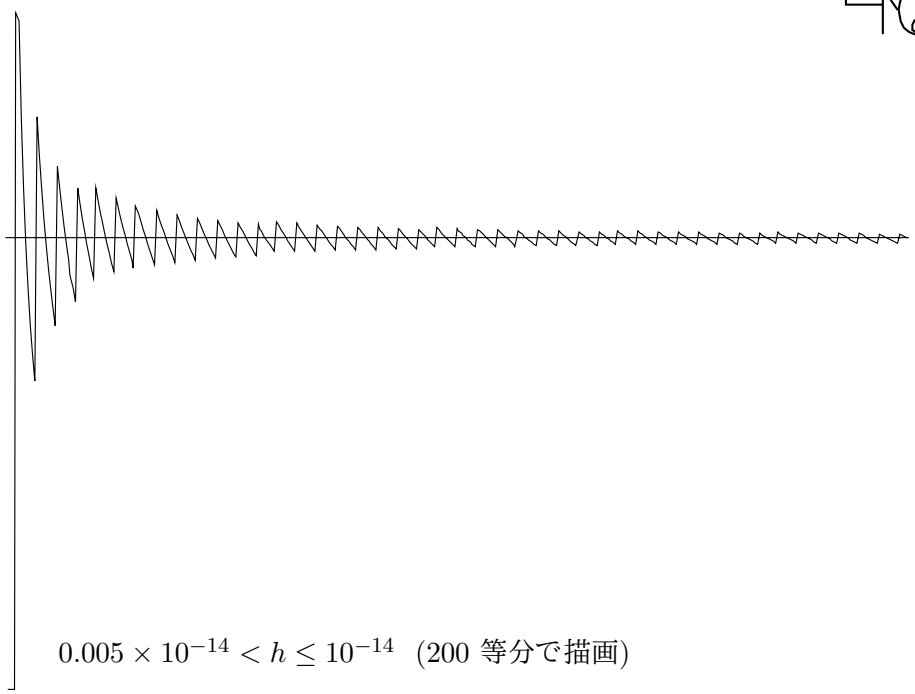
心配は現実になりそう.



$10^{-14} \leq h \leq 2 \times 10^{-14}$ (200 等分で描画)

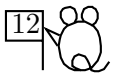
次はどうなる？

じゃじゃじゃーん!



$0.005 \times 10^{-14} < h \leq 10^{-14}$ (200 等分で描画)

桁落ち： 浮動小数の加減算は可換ではない!



上で観察したのは, **桁落ち**という実数計算における恐ろしい現象.
計算機が扱う小数が有限の長さに丸められることから起こる.

固定小数点表現： 0.00001234
浮動小数点表現： 0.1234×10^{-4}

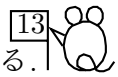
後者の方が無駄が少ないので, 計算機内部では二進浮動小数点数表現を用いる.
これにより, どんな大きさの数でも仮数部の有効桁数をほぼ一定に保てる.
$\Longrightarrow$ 掛け算には便利. しかし, 足し算には?

有効桁数が十進 16 桁として, 次の計算を考察する

$$\begin{array}{rl} & 1.000000000000000 \\ + & 0.000000000000012345678901 2345 \\ \hline & 1.000000000000123 \end{array} \qquad \begin{array}{rl} & 1.000000000000123 \\ - & 1.000000000000000 \\ \hline & 0.000000000000123 \end{array}$$

この順序で計算すると, 有効桁数が 12 桁分も失われてしまった.
一般に, h が小さいときに $1+h$ という計算をすると,
h が持つの情報の 10^{-16} 以下の部分は捨てられる (**丸め誤差**の発生).
その結果から 1 を引いても, 完全には h に戻らない.
実用計算では桁落ちを防ぐため最大限努力しないと, 結果が信用できなくなる.
(桁が落ちたせいで橋が落ちるかもしれない. (^^;)

情報落ち： 浮動小数の加減算は結合法則を満たさない!



最初の加算の段階で二つ目の数の情報が既に 12 桁分も失われている.
それが最終結果なら, 特に問題は無い. では加算だけなら常に問題は無いのか？

```
    1.000000000000000
    0.000000000000123456789012345
    0.000000000000000478901234562345
    0.000000000000000008901234562345678
+   0.000000000000000009012345623456678
-----------------------------------------
    1.000000000000123
```

上から順に加えれば, 結果は前と変わらない. しかし, 下の4 個を先に加えると

```
    0.000000000000123456789012345
    0.000000000000000478901234562345
    0.000000000000000008901234562345678
+   0.000000000000000009012345623456678
-----------------------------------------
    0.000000000000124114826048765
```

となり, これを1 番上に加えると, 結果は 1.000000000000124 となる.
小さい数がたくさんあると結果はもっと深刻に変わる.

IEEE 754 規格　　最も普及している二進浮動小数点数の表示法

☆ **単精度**　4 バイト $=32$ ビット (指数部 8 + 仮数部 24)

$$\pm\Big(\frac{1}{2}+\frac{a_2}{2^2}+\cdots+\frac{a_{24}}{2^{24}}\Big)\times 2^e,\qquad -126\le e\le 128$$

☆ **倍精度**　8 バイト $=64$ ビット (指数部 11 + 仮数部 53)

$$\pm\Big(\frac{1}{2}+\frac{a_2}{2^2}+\cdots+\frac{a_{53}}{2^{53}}\Big)\times 2^e,\qquad -1022\le e\le 1024$$

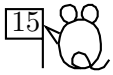
指数を調節して, 小数点第1 位が 0 でないようにすると,
二進法では 1 に決まるので書かずに済む.
その分を全体の符号を表す 01 の情報に回す (**規格化浮動小数点数**).

※ 従って規格化されているときは，符号+ 指数部で 12 ビット，
　仮数部で 52 ビットを使用.

これらのビットデータが計算機内部でどのように置かれているかは
かなりマニアックな話になる.
しかし, 本気でプログラミングをやろうとすると, 常に気にする必要がある.

$\Longrightarrow$ ビッグエンディアンとリトゥルエンディアン

ビッグエンディアンとリトゥルエンディアン

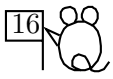


endian とは, 卵の太い方, 細い方のどちらを好むかという話で,
ガリバー旅行記に, これが元で戦争をしている国の話が登場する.
普通の PC やワークステーションでは，メモリーの一つのアドレスに
1 バイトずつ格納される.
複数バイトより成るデータは, メモリーの下位から上位にどのような順番で
並んでいるか:

例: 十六進数 897EB85A の場合
big endian: 上位から順に1 バイトずつメモリーの下位の方から並ぶ.
89 7E B8 5A
little endian: データの下位がメモリーの下位と一致.
5A B8 7E 89
インテルのプロセッサは little endian,
Power PC はどちらか選べる. デフォールトはどうなっているか調べてみよ.

Q 昔は情報科学科でも，UltraSparc とか PA-RISC とか，
さまざまな CPU に学生が触ることができたが，今は上の二つくらいに
なってしまった. 寂しい.

Intel Pentium 機でメモリーの内部を覗いてみた結果 I



例 十進法の 1.0 は浮動小数でも誤差無しで格納される：
$0.\underbrace{100\cdots0}_{53\text{ ビット}}\times 2^1$ 指数にバイアス 1022 が足され 1023=1024-1 (3FF) となり,
仮数部の最初の1 ビットが省略され, $\underbrace{\texttt{3F F}}_{\text{符号}+\text{指数}}\texttt{0 00 00 00 00 00 00}$ となる.

例 十進法の $0.2=\dfrac{1}{5}$ は二進法では無限循環小数 $\dfrac{1}{101}=0.00110011\cdots$
これは倍精度二進小数で有限小数に丸められるが
$0.\underbrace{110011001100\cdots11001}_{53\text{ ビット}}\times 2^{-2}$ に切捨てられるのではなく
$0.\underbrace{110011001100\cdots11010}_{53\text{ ビット}}\times 2^{-2}$ に四捨五入され, 最初の1 ビットが省略され,
指数 $+1022=1020=1024-4$ (3FC), $\underbrace{\texttt{3F C}}_{\text{符号}+\text{指数}}\texttt{9 99 99 99 99 99 9A}$ となる.
実際にメモリーを覗くと, これが完全に逆順に置かれている (little endian):
`2.0000000000000000e-01 = 9A 99 99 99 99 99 C9 3F`

負数の例 十進法の -0.2 だと, 先頭に符号ビットが立ち,
`BF C9 99 99 99 99 99 9A` となるはずだが, メモリーでは
`-2.0000000000000000e-01 = 9A 99 99 99 99 99 C9 BF`

Q 整数の場合は負数を表すのに, 先頭の1 ビットを立てるだけでなく,
いわゆる**2 の補数表現**を用いる： 例えば $-1\leftrightarrow$ `FF FF FF FF` $=2^{32}-1$
しかし, 浮動小数の指数部では, 負数の表現に2 の補数は用いず,
最小の負数が0 となるようにバイアスを一斉に加えて全体をシフトしている.

倍精度浮動小数で表される**最大の正数**は

$$\left(\frac{1}{2}+\frac{1}{2^2}+\cdots+\frac{1}{2^{53}}\right)\times 2^{1023} \fallingdotseq 1.797\cdots\times 10^{308}$$

これより大きくなると, 実数型での**オーバーフロー**となる.

倍精度浮動小数で表される**最小の正数**は (右端はビッグエンディアン表記)

$$\frac{1}{2}\times 2^{-1021}=2^{-1022}\fallingdotseq 2.225\cdots\times 10^{-308} \qquad 00\,10\,00\,00\,00\,00\,00\,00$$

この次の $\frac{1}{2}\times 2^{-1022}$ は表示がすべて 0 となり，真の 0 と区別できないので**漸近的アンダーフロー**となる.

しかし, 指数が最小値 -1022 のときは**規格化を止め仮数部 52 ビットをすべて書くことにする**と, 表せる最小値はもう少し増え，この続きは

$$\frac{1}{2}\times 2^{-1022}=2^{-1023}\fallingdotseq 1.1125\cdots\times 10^{-308} \qquad 00\,08\,00\,00\,00\,00\,00\,00$$

$$\vdots$$

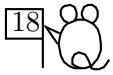
$$\varepsilon := \frac{1}{2^{52}}\times 2^{-1022}=2^{-1074}\fallingdotseq 4.940\cdots\times 10^{-324} \qquad 00\,00\,00\,00\,00\,00\,00\,01$$

⇖ **計算機イプシロン**

計算機で $h\to 0$ の極限を計算しようと思っても, $h\geq\varepsilon$ の範囲でしか動けない.
$h<\varepsilon$ となって, 0 と同一視されてしまった状態が真の**アンダーフロー**

∞ ここでの話はあくまで“ 通常の変数を用いた通常の計算 ”の説明.
実は, 多倍長演算の考えは浮動小数点数にも適用できる.
そういうものを使うと, 時間とメモリーが許す限りいくらでも
大きな, あるいは小さな数を扱えるようになる.
cf. 金田康正氏による π の 1 兆桁計算.

Intel Pentium 機でメモリーの内部を覗いてみた結果 II

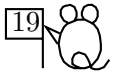


CPU: Pentium 機 で 1 を 2 で次々に割って行き,
アンダーフローを確認

```
1.000000000000000e+00 = 00 00 00 00 00 00 F0 3F
5.000000000000000e-01 = 00 00 00 00 00 00 E0 3F
2.500000000000000e-01 = 00 00 00 00 00 00 D0 3F
1.250000000000000e-01 = 00 00 00 00 00 00 C0 3F
6.250000000000000e-02 = 00 00 00 00 00 00 B0 3F
3.125000000000000e-02 = 00 00 00 00 00 00 A0 3F
1.562500000000000e-02 = 00 00 00 00 00 00 90 3F
.........
4.450147717014403e-308 = 00 00 00 00 00 00 20 00
2.225073858507201e-308 = 00 00 00 00 00 00 10 00
1.112536929253601e-308 = 00 00 00 00 00 00 08 00 ←ここから非規格化
5.562684646268003e-309 = 00 00 00 00 00 00 04 00
.........
1.976262583364986e-323 = 04 00 00 00 00 00 00 00
9.881312916824931e-324 = 02 00 00 00 00 00 00 00
4.940656458412465e-324 = 01 00 00 00 00 00 00 00 ←計算機イプシロン
0.000000000000000e+00 = 00 00 00 00 00 00 00 00
```

Q この表の一行目は倍精度小数とみなされた1 の内部表現である.
整数の1 とは計算機内部での表現 (データ構造) が異なることに注意!

丸め誤差に関するもう一つの有名な例



★ $\left(1+\frac{1}{n}\right)^n$ は $n \to \infty$ のとき収束するか?

十桁の電卓による実験結果:

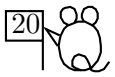
n	$(1+1/n)^n$
10	2.59374246
100	2.704813829
1,000	2.716923932
10,000	2.718145927
100,000	2.718268237
1,000,000	2.718280469
10,000,000	2.718281693
100,000,000	2.718281815
1,000,000,000	2.718281827
2,000,000,000	2.718281828

この結果は非常にもっともらしい.
うっかりすると, 高校の教科書にこのまま載っていたりする.

しかし, 君達はだまされている!
計算機で本当に掛算を実行するとこうはならない:

計算機実験の結果:

n	$(1+1/n)^n$
10	2.593742460100002
100	2.704813829421529
1,000	2.716923932235598
10,000	2.718145926824898
100,000	2.718268237192295
1,000,000	2.718280469095936
10,000,000	2.718281694132010
100,000,000	2.718281798346360
1,000,000,000	2.718282052011900
2,000,000,000	2.718282052691296



理論値は常に e の真の値より小さいはずだが, 最後の方で逆転している.

$$\left(1+\frac{1}{n}+\varepsilon\right)^n = \left(1+\frac{1}{n}\right)^n + n\left(1+\frac{1}{n}\right)^{n-1}\varepsilon + \cdots \quad \text{(二項定理)}$$

ここで, $\varepsilon \doteqdot 10^{-17}$, $n = 10^9$ とすると, 誤差項は $\doteqdot 10^{-7}$ となる.
もし函数電卓が $\varepsilon \doteqdot 10^{-11}$ で計算していたら, 誤差は 10^{-2} 程度になるはず!

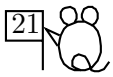
Taylor 展開による, 理論誤差の計算:

$$\left(1+\frac{1}{n}\right)^n = \exp\left[n\log\left(1+\frac{1}{n}\right)\right] = \exp\left[n\left(\frac{1}{n}-\frac{1}{2n^2}+\cdots\right)\right] = \exp\left(1-\frac{1}{2n}+\cdots\right)$$

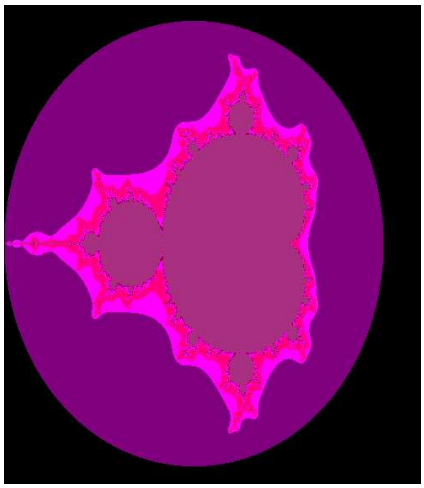
$$= e\cdot\exp\left(-\frac{1}{2n}+\cdots\right) = e\cdot\left(1-\frac{1}{2n}+\cdots\right) = e-\frac{e}{2n}+\cdots$$

電卓の計算結果は, これに近い値となっている. 何故だ! ($\Longleftarrow$ 挑戦課題 (^^;))

計算機イプシロンを目で見よう！



右図はフラクタルの一種, **マンデルブロート 集合**である.
(竹尾研より 拝借 (^^;).



作り 方: 複素平面で, 各 $c \in \boldsymbol{C}$ に対し
原点から 出発して $z \mapsto z^2 + c$ という 写像を
繰り 返して 適用したとき,
いつまでも $|z| \leq 2$ に 留まるとき
c はマンデルブロート 集合に 属するとし,
その点 c を 黒く 塗る.

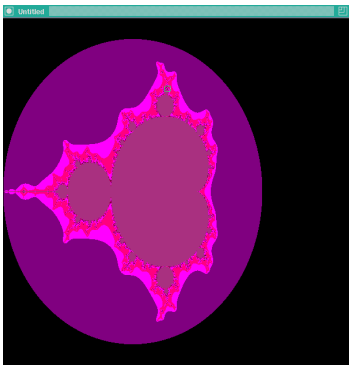
いつか $|z| > 2$ に 飛び出すときは
それまでにさまよった 回数で 色分けする.

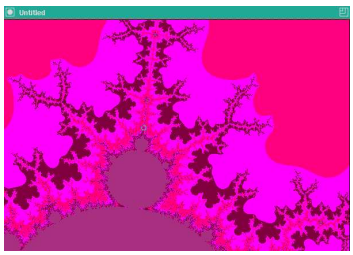
この図形の 部分を 拡大すると,
同じ ような 形の 複雑な 図形が 繰り 返し 現れ,
数学的には どこまでも 続く.

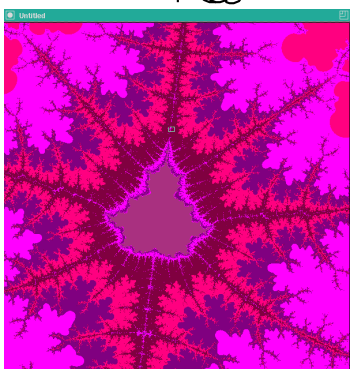
計算機で 実際に 拡大を 続けると どうなるか?

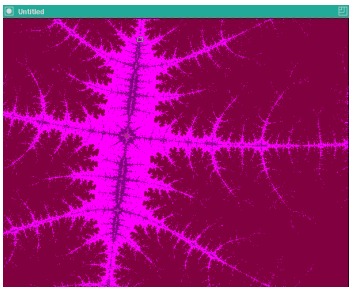
実　演　！

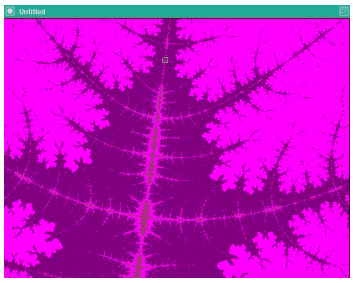
ちょっと分かりにくいが, 点線で囲まれた領域を次に拡大している：

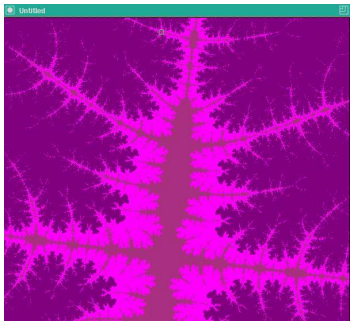


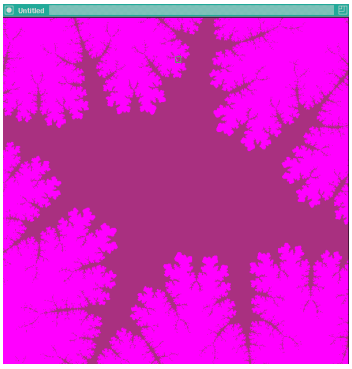


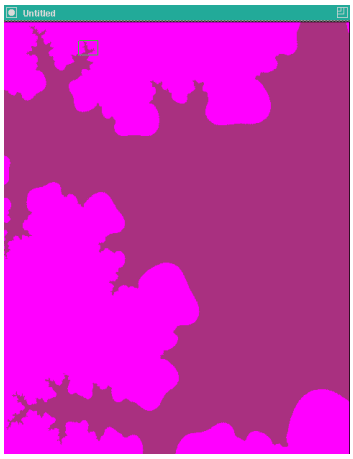


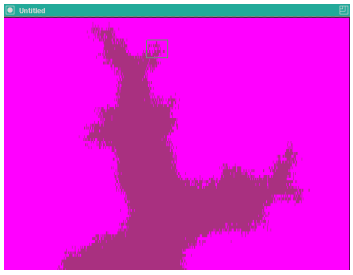






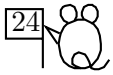






最後はモザイク模様になった！
この長方形の一辺が計算機イプシロンを表している．
(縦と横の長さが異なるのは, 途中の拡大率が異なったため.)

本日のまとめ

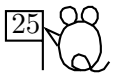


1 コンピュータの内部では, すべてが 0 と 1 の有限列で表される.

2 整数も小数も決まった長さの記憶領域に収められるので,
その限界を越えるとオーバーフローやアンダーフローが生じる.

3 小数は有限で打ち切られる.
そのため桁落ち, 情報落ちが生じ,
通常の四則演算の公理は必ずしも満たされなくなる.

本日の講義内容の自習課題



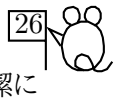
(1) 桁落ちと情報落ちを体験してみましょう.

(2) 5 階の計算機室の Intel CPU がビッグエンディアンか
リトルエンディアンかを調べるプログラムを使ってみましょう.
その後で，PowerPC をエミュレートしたときどうなるかを
調べてみましょう.

(3) 浮動小数の内部表現を調べるプログラムを使ってみましょう.

(4) オーバーフローとアンダーフローを体験してみましょう.

(5) 計算機 ε を可視化するプログラムを実行してみましょう.



問題 1.1 ビッグエンディアンとリトゥルエンディアンについて簡潔に説明せよ.

問題 1.2 リトゥルエンディアンのインテル CPU のコンピュータで倍長整数 (long) 1025, 単精度浮動小数 (float) `-0.1000000`, 倍精度浮動小数 (double) `1.25000000000000` が格納されたメモリーの内容の十六進表記としてそれぞれ最もふさわしいものを次のうちから選べ.

(1) `CD CC CC 3D` (2) `CD CC CC BD` (3) `BD CC CC CD`
(4) `01 04 00 00` (5) `00 00 10 25` (6) `00 00 01 04`
(7) `00 00 25 10` (8) `FF FB FF FF` (9) `00 00 F4 3F`
(10) `00 00 00 00 00 00 F4 3F`
(11) `00 00 00 00 00 40 5F 40`

問題 1.3 リトゥルエンディアンのインテル CPU のコンピュータでメモリー内に存在した4 バイトのデータ `CD CC 4C 3E` に該当するのは次のうちどれか?

(1) 文字列 `DCK>`
(2) 単精度浮動小数 `0.2`
(3) 倍長整数 `123,000,395,296`
(4) 倍長整数 `-123,000,395,297`

問題 1.4 オーバーフローとアンダーフローについて説明せよ．
また計算機イプシロン (machine epsilon) とは何か？

問題 1-5 (1) 符号無し倍長整数 (unsigned long) で $n!$ を計算する関数のプログラムを C 言語で書け．
(2) このプログラムが正しい値を与える範囲を見積もれ．
[ヒント： Taylor 展開から得られる $\frac{n^n}{n!} \leq e^n$ などを用いて $n!$ を見積もれ．計算機が使えない状況では限界の n を正確に出せなくてもよい．実は計算機を使っても 31 とか答えた人が沢山居たので，こういう練習も必要です．]

問題 1-6 単精度で $\sum_{n=1}^{N} \frac{1}{n}$ を頭から計算して行ったとき, 和の値 S が変化しなくなる最小の N の値を N_0 とする．このとき，N_0 から 1 まで n を逆に動かして上の和を計算したものの値は，S より大きいか，それとも小さいか，それとも同じか？ 理由とともに答えよ．

問題 1-7 次のようなデータの計算機内部での値を十六進数で表現せよ．

(1) 倍長整数 (long,32 ビット) 2011
(2) 倍長整数 (long,32 ビット) -2011
(3) 単精度浮動小数 4.0
(4) 倍精度浮動小数 2.0
(5) 文字列 `"I love Ochanomizu."`
[ヒント： ピリオドのアスキーコード `0x2E` は試験時にも与える．半角空白のコードとアルファベットの大文字小文字の開始コードくらいは記憶せよ．]