数値計算の誤差

柏木 雅英
kashi@waseda.jp
http://verifiedby.me/

早稲田大学 基幹理工学部 応用数理学科

整数型のオーバーフロー (wrap around)

- 整数型のオーバーフロー (wrap around という) は、浮動小数点数のように無限大のような特殊な数になるわけではなく、黙って異常な結果になるので注意が必要。

int の wrap around

```c
#include <stdio.h>
#include <limits.h>

int main()
{
    signed int x;
    int i;

    x = INT_MAX - 2;
    for (i=0; i<6; i++) {
        printf("%d\n", x);
        x++;
    }

    printf("\n");

    x = INT_MIN + 2;
    for (i=0; i<6; i++) {
        printf("%d\n", x);
        x--;
    }
}
```

```
2147483645
2147483646
2147483647
-2147483648
-2147483647
-2147483646

-2147483646
-2147483647
-2147483648
2147483647
2147483646
2147483645
```

unsigned int の wrap around

```c
#include <stdio.h>
#include <limits.h>

int main()
{
    unsigned int x;
    int i;

    x = UINT_MAX - 2;
    for (i=0; i<6; i++) {
        printf("%u\n", x);
        x++;
    }

    printf("\n");

    x = 0 + 2;
    for (i=0; i<6; i++) {
        printf("%u\n", x);
        x--;
    }
}
```

```
4294967293
4294967294
4294967295
0
1
2

2
1
0
4294967295
4294967294
4294967293
```

フェルマーの最終定理の反例!?

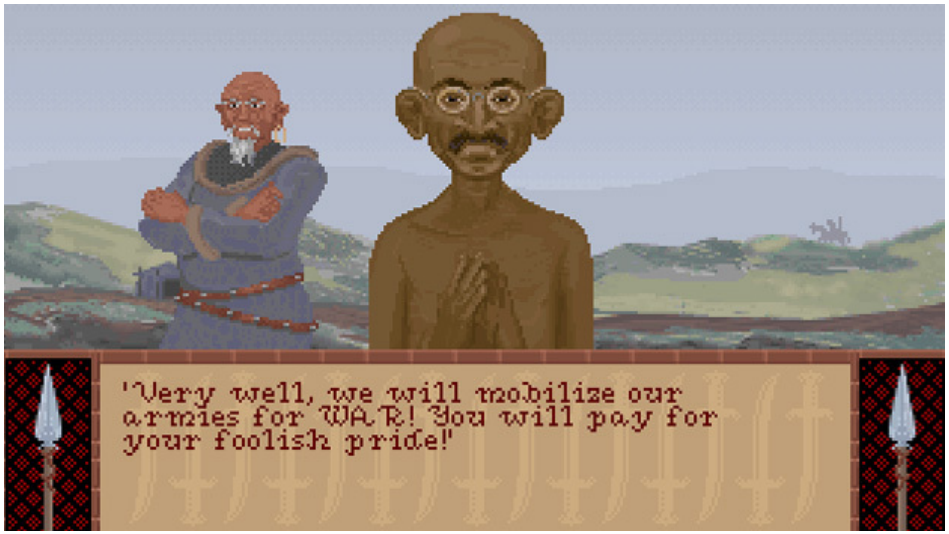
```
#include <stdio.h>

int main()
{
    int a, b, c;

    for (c=2; c<=100000; c++) {
        for (a=1; a<c; a++) {
            for (b=a; b<c; b++) {
                if (a*a*a + b*b*b == c*c*c) {
                    printf("%d %d %d\n", a, b, c);
                    printf("%d %d %d\n", a*a*a, b*b*b, c*c*c);
                }
            }
        }
    }
}
```

$$139^3 + 954^3 = 2115^3 \quad (?)$$

核ガンジー



Civilization というゲームの初期バージョンで、平和的なキャラクターのはずのガンジーが、ある条件で突如凶暴になり核攻撃を仕掛けてくるというバグがあった。リーダーの行動を決めるパラメータの一つに攻撃性があり、$0-255$ の数値で表現されていた。ガンジーは最も低い数値の 1 が与えられていた。しかし、ある文明が民主主義を採用したとき相手の攻撃性を -2 するという仕様になっていて、このときガンジーの攻撃性が wrap around により 255 となり、最悪の攻撃性を持つキャラクターとなってしまうというバグだった。このバグは大変面白かったため、後のバージョンにもわざとそのまま引き継がれたという。この話は非常に有名になったが、2020 年のシド・マイヤーの著書で、この話は都市伝説で実際にはそのようなバグは存在しなかったことが明かされた。

浮動小数点数のオーバーフロー

```c
#include <stdio.h>

int main()
{
    double x;
    int i;

    x = 3.14159265358979;
    printf("%.17g\n", x);

    for (i=0; i<310; i++) {
        x *= 10.;
        printf("%.17g\n", x);
    }
}
```

```
3.14159265358979
31.415926535897899
314.15926535897898
3141.5926535897897
31415.926535897896
314159.26535897894
3141592.6535897893
31415926.535897892
314159265.35897893
3141592653.5897894
31415926535.897896
314159265358.97894
3141592653589.7896
31415926535897.895
314159265358978.94
3141592653589789.5
31415926535897896
3.1415926535897894e+17
(omitted)
3.1415926535897901e+300
3.1415926535897899e+301
3.1415926535897899e+302
3.1415926535897901e+303
3.1415926535897904e+304
3.1415926535897906e+305
3.1415926535897906e+306
3.1415926535897908e+307
inf
inf
inf
```

浮動小数点数のアンダーフロー

```c
#include <stdio.h>

int main()
{
    double x;
    int i;

    x = 3.14159265358979;
    printf("%.17g\n", x);

    for (i=0; i<327; i++) {
        x /= 10.;
        printf("%.17g\n", x);
    }
}
```

非正規化数の領域で精度が失われる様子がよく分かる。

```
3.14159265358979
0.31415926535897898
0.031415926535897899
0.0031415926535897898
0.00031415926535897898
3.1415926535897901e-05
(omitted)
3.14159265358979004e-305
3.14159265358979003e-306
3.14159265358979003e-307
3.14159265358979004e-308
3.14159265358979009e-309
3.14159265358980057e-310
3.14159265358980954e-311
3.14159265358084718e-312
3.14159265358084718e-313
3.14159265378609 8e-314
3.14159265329203 24e-315
3.14159264341071 95e-316
3.14159249510102 57e-317
3.14159051892844 23e-318
3.14156581564615 03e-319
3.14176344190448 68e-320
3.14225750755032 8e-321
3.16202013338397 79e-322
2.9643938750474793e-323
4.9406564584124654e-324
0
0
0
```

丸め誤差

- 計算結果などの数値をメモリに格納するとき、仮数部の長さに限界があるため (倍精度なら 53bit、単精度なら 24bit)、はみ出た部分は切り捨てるか切り上げるかするしかない。ここで発生する誤差を丸め誤差という。
- 切り捨てるか切り上げるかは、誤差の小さくなる方を採用する (10 進数で言うところの四捨五入のようなもの)。2 進数なので 0 捨 1 入。
- (偶数丸め) 切り捨てと切り上げの誤差が完全に等しいとき、すなわち、格納したい数が 2 つの隣り合った浮動小数点数のちょうど中間だったとき、仮数部の末尾の bit が 0 である方に丸めるというルールになっている。これを偶数丸めという。

絶対誤差, 相対誤差, machine epsilon

ある値 x になんらかの誤差が入って $\tilde{x}$ になったとき、

$$|x - \tilde{x}|$$

を絶対誤差、

$$\frac{|x - \tilde{x}|}{|x|} \quad \text{あるいは} \quad \frac{|x - \tilde{x}|}{|\tilde{x}|}$$

を相対誤差という。浮動小数点形式は、相対誤差をほぼ一定値に保つように設計されている。

machine epsilon

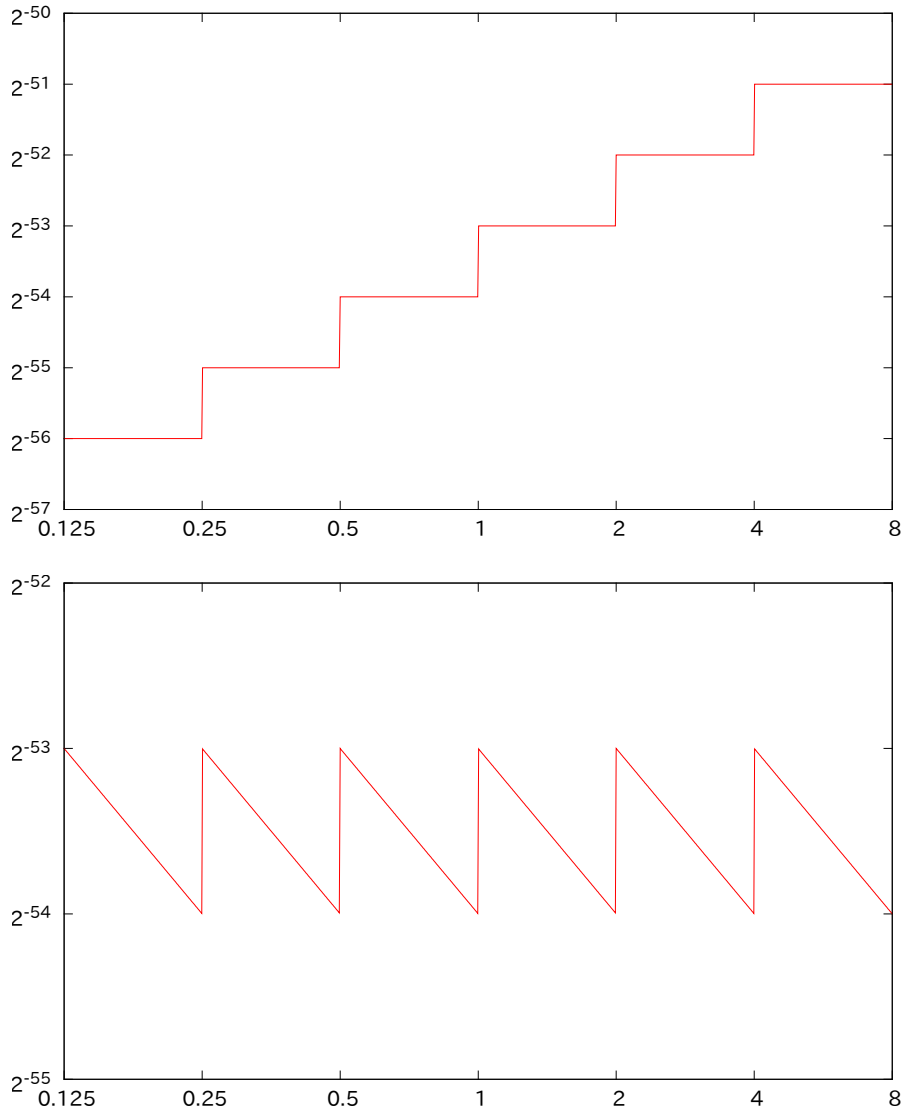
ある浮動小数点数の体系において、

$$(1 \text{ より大きな最小の浮動小数点数}) - 1$$

を、machine epsilon という。

相対誤差の大きさの目安として使われる。IEEE 754 倍精度では $2^{-52} \simeq 2.22 \times 10^{-16}$、単精度では $2^{-23} \simeq 1.19 \times 10^{-7}$。

絶対誤差と相対誤差


2-50
2-51
2-52
2-53
2-54
2-55
2-56
2-57
0.125
0.25
0.5
1
2
4
8
2-52
2-53
2-54
2-55
0.125
0.25
0.5
1
2
4
8

$\frac{1}{10}$ はどのように格納されるか

$\frac{1}{10} = 0.1$ は、10 進数だと有限小数だが、2 進数だと無限小数。

$$0.1_{(10)} = 0.0001100110011001100\cdots_{(2)}$$

$1.10011001100110011001100110011001100110011001100110011001|1001100\cdots \times 2^{-4}$
のうち先頭の 53bit が格納可能なので、54bit 目を 0 捨 1 入して、
$1.10011001100110011001100110011001100110011001100110011010 \times 2^{-4}$
を格納する。$-4 + 1023 = 1019$ で、$1019_{(10)} = 01111111011_{(2)}$ なので、
$0|01111111011|1001100110011001100110011001100110011001100110011010$
のような bit 列で格納される。この数は、10 進数に直すと

$$0.1000000000000000055511151231257827021181583404541015625$$

で、0.1 より少し大きい数になっている。

10 進数に慣れている我々は 0.1 はきれいな割り切れる数という思い込みがあるので、0.1 に誤差が混入していることによって引き起こされる事故はとても多い。

0.1 が正確に表現できないので・・・

```
#include <stdio.h>

int main()
{
    double x;

    for (x = 0; x <= 0.3; x += 0.1) {
        printf("%f\n", x);
    }
}
```

```
0.000000
0.100000
0.200000
```

$0.1 + 0.1 + 0.1$ が 0.3 を上回ってしまい、最後のループが実行されない。

積み残し

またの名を「焼け石に水」。大きな値に小さな値を加えたときに小さな値の情報の大半、または全てが失われてしまうような現象を言う。
例えば、

$$(3.14159265358979 + 10^{10}) + (-10^{10})$$
$$3.14159265358979 + (10^{10} + (-10^{10}))$$

をそれぞれ計算すると、

3.1415920257568359
3.14159265358979

となり、前者は 7 桁程度しか情報が残っていないことが分かる。前者の第一項は

$$3.14159265358979 + 10^{10} = 10000000003.14159265358979$$

となるが、有効数字 16 桁程度しか無いのでそこで丸められてしまうことによる。浮動小数点数で加法の結合法則が成立しない例にもなっている。

2 次方程式の丸め誤差 (1)

2 次方程式

2 次方程式 $ax^2 + bx + c = 0$ の解の公式は、

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

大きな誤差が見られる例

$$a = 1,\ b = 10^{15},\ c = 10^{14}$$

のときの大きい方の解を計算する。

- (解の公式) $\dfrac{-b + \sqrt{b^2 - 4ac}}{2a} = -0.125$
- (解の公式の分子を有理化)
  $\dfrac{2c}{-b - \sqrt{b^2 - 4ac}} = -0.10000000000000002$

2 次方程式の丸め誤差 (2)

桁落ち

非常に近い 2 数同士の減算で、有効数字が極端に減少してしまうこと。

$$\begin{array}{rr} & 3.141592X \\ - & 3.141591X \\ \hline & 0.000001X \end{array}$$

7 桁まで正しく 8 桁目が怪しい 2 数の減算で、1 桁しか合ってない数値が現れる例。実は減算そのものには罪はなく (むしろ非常に近い数同士の減算では全く誤差は発生しない)、元々数値に入っていた誤差が顕在化してしまう現象である。

前の 2 次方程式の例だと、a, b, c の大きさの関係で、$b^2 - 4ac$ はほぼ b^2、$\sqrt{b^2 - 4ac}$ はほぼ b なので、$-b + \sqrt{b^2 - 4ac}$ の部分で桁落ちが発生している。

2 次方程式の丸め誤差 (3)

2 次方程式の解は、次のように計算するとよいとされている。

1. まず解の片方を計算する。b の正負で場合分けし $b \geq 0$ なら

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

を、$b < 0$ なら

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

を計算する (桁落ちの起きない方を計算) 。

2. 他方の解を、解と係数の関係を使って

$$x_2 = \frac{c}{ax_1}$$

で計算する。

連立一次方程式の誤差

連立一次方程式

$$\begin{pmatrix} 64919121 & -159018721 \\ 41869520.5 & -102558961 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

真の解

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 205117922 \\ 83739041 \end{pmatrix}$$

ガウスの消去法で計算した解

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 106018308.0071325 \\ 43281793.001783125 \end{pmatrix}$$

間違った解の残差を計算すると $\cdots$

$$\begin{pmatrix} 64919121 & -159018721 \\ 41869520.5 & -102558961 \end{pmatrix} \begin{pmatrix} 106018308.0071325 \\ 43281793.001783125 \end{pmatrix} - \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} ??$$

残差が小さければ誤差は小さい?

連立一次方程式

$$\begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix}$$

真の解

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

真の解に近い近似解 $\begin{pmatrix} 0.999 \\ -1.001 \end{pmatrix}$ に対する残差

$$\begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix} \begin{pmatrix} 0.999 \\ -1.001 \end{pmatrix} - \begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix} = \begin{pmatrix} -0.001343 \\ -0.001572 \end{pmatrix}$$

でたらめな近似解 $\begin{pmatrix} 0.341 \\ -0.087 \end{pmatrix}$ に対する残差

$$\begin{pmatrix} 0.780 & 0.563 \\ 0.913 & 0.659 \end{pmatrix} \begin{pmatrix} 0.341 \\ -0.087 \end{pmatrix} - \begin{pmatrix} 0.217 \\ 0.254 \end{pmatrix} = \begin{pmatrix} -0.000001 \\ 0 \end{pmatrix} ??$$

Rump の例題

Rump の例題 (改)

$a = 77617$、$b = 33096$ に対して、以下の値を計算せよ。

$$(333.75 - a^2)b^6 + a^2(11a^2b^2 - 121b^4 - 2) + 5.5b^8 + a/(2b)$$

計算結果

1.17260396480560302734375 (float)
1.1726039400531786949244406059733592 (double)
1.1726039400531786318588349045201801 (double-double)
1.1726039400531786318588349045201838 (mpfr113)
1.1726039400531786318588349045201838 (binary128)
-0.827396059946821368141165095479816291 (mpfr150) ←true value

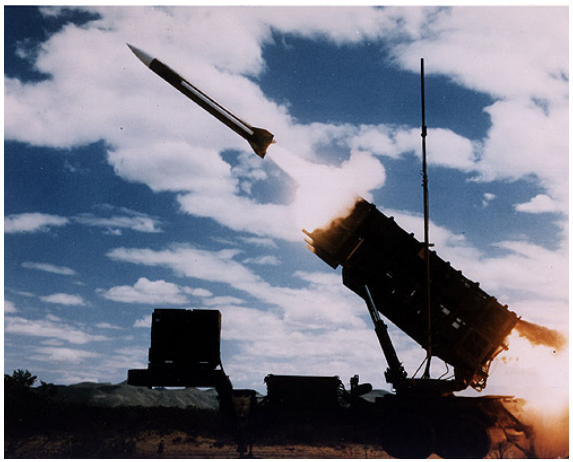
float(仮数部 23bit) から binary128(仮数部 113bit) まで、その精度に応じて計算できているように見えて実はデタラメで、仮数部 150bit でやっと本当の値が見える。

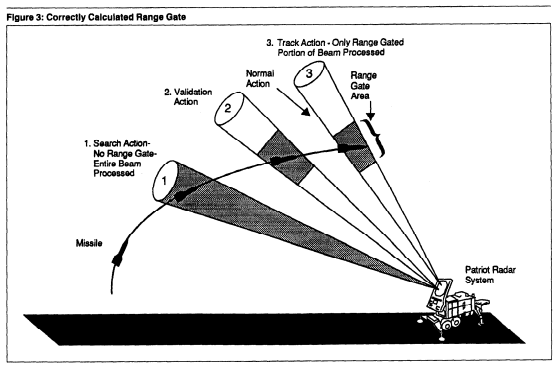
湾岸戦争におけるパトリオットミサイルの事故

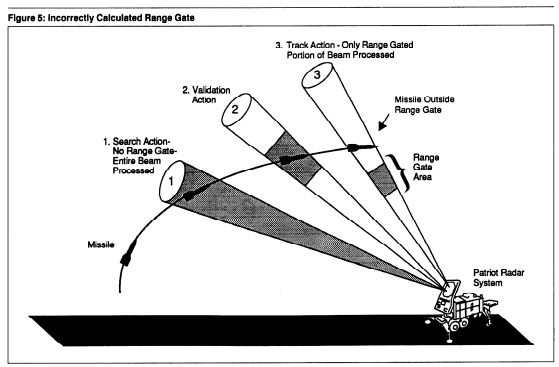
http://www.sydrose.com/case100/298/
https://www.gao.gov/assets/220/215614.pdf

湾岸戦争中、イラク軍のスカッド・ミサイル (Scud Missile) の迎撃のため、アメリカ軍のパトリオット・ミサイル (Patriot Missile) が発射されたが、その内部計算機のわずかな誤差 (0.1 秒を正確に表現できなかったことによる) からスカッド・ミサイルを捉えることが出来ず迎撃は失敗した。

- 発生日時 1991 年 2 月 25 日
- 発生場所 サウジアラビア、Dharan
- 死者 28 名、負傷者約 100 名

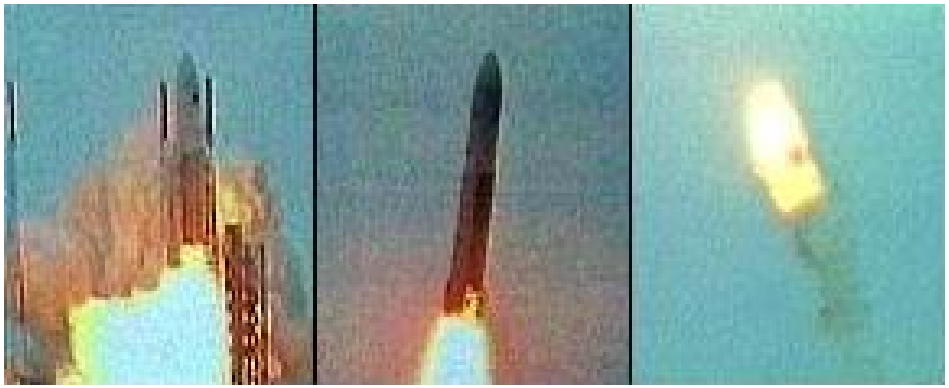






アリアン5型ロケット爆発事故

http://www.sydrose.com/case100/284/
http://www.math.umn.edu/~arnold/disasters/ariane.html



爆発事故

1996 年 6 月 4 日、欧州宇宙機関 (European Space Agency) が新たに開発した無人ロケット、アリアン 5 (Ariane 5) の最初の打ち上げがフランス領ギアナのクルーで行われたが、発射のわずか 40 秒後に進路を大きく逸れ、空中で爆発した。アリアン 5 の開発には 10 年の歳月と 70 億ドルの費用が投じられていた。

原因

慣性システムのソフトウェアに問題があった。ロケットの水平速度に関する数値を 64bit の浮動小数点数から 16bit の符号付き整数に変換する部分で 16bit 符号付き整数の最大値 32767 を超えてしまい変換に失敗、正確なロケットの姿勢データを得ることが出来なくなってしまった。

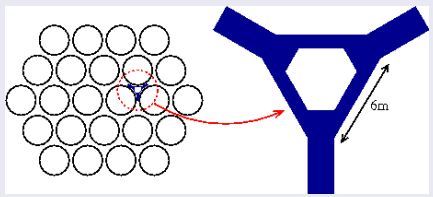
シュライプナー A 海上プラットフォーム沈没事故

http://www.shippai.org/fkd/cf/CA0000299.html
http://www.math.umn.edu/~arnold/disasters/sleipner.html

沈没事故

北海で石油とガスを産出するための海上プラットフォーム Sleipner A は、1991 年 8 月 23 日、ノルウェー沖での建設中に水中の基礎構造物に破損が生じて浸水し、崩壊、沈没した。

原因



崩壊の原因は、基礎構造物の 24 本の柱の間にある、tricell と呼ばれるコンクリートの構造物が、強度の不足により破断したためであった。有限要素法での計算の精度に問題があって圧力が実際の 47%と低く計算され、それが設計の強度不足を招いてしまった。

ボーイング 787 の 248 日問題

2015 年 4 月 30 日、アメリカ連邦航空局 (FAA) はボーイング 787 の電源制御システムのソフトウェアに問題があり、248 日間継続してシステムを稼働させ続けた場合、突然電源が喪失し機体制御が失われる恐れがあるとして、電気系統を定期的に再起動をする国内各航空会社に通達を出した。システム内に $\frac{1}{100}$ 秒毎にカウントアップするするタイマーがあり、248.55 日間連続稼働すると `signed int` の限界である 2^{31} を超えてしまうためと見られている。

2015 年 5 月 4 日、欧州航空安全機関 (EASA) もこれに続き、ボーイング社は年末までにソフトウェアの改修を行った。

数値解析

代数的に解を得ることが不可能な解析学上の問題を、数値を用いて近似的に解く。

自然現象

$\Downarrow$ モデル化

微分方程式

$\Downarrow$ 離散化　　(離散化誤差が混入)

(有限次元) 方程式

$\Downarrow$ 数値計算　　(丸め誤差, 打ち切り誤差が混入)

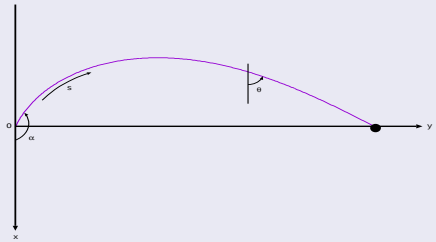
数値解

数値解析の誤差の例

自然現象 (釣り竿の撓み)

⇓ モデル化

微分方程式 $\dfrac{d^2\theta}{ds^2} = K\sin\theta, \quad \theta(0) = \alpha, \quad \dfrac{d\theta}{ds}(1) = 0$



⇓ 離散化 (離散化誤差が混入)

(有限次元) 方程式

$$\theta_{k+1} = \theta_k + \phi_k h \qquad \theta_0 = \alpha$$
$$\phi_{k+1} = \phi_k + K\sin\theta_k h \qquad \phi_n = 0$$

⇓ 数値計算 (丸め誤差, 打ち切り誤差が混入)

数値解

k	θ	ϕ
0	2.7925268031909272	-3.272746776503749
1	2.4652521255405522	-3.1359387191734815
⋮		
10	0.88043670714284783	1.6653345369377348e-16