数値解析　(塩田)　2013 年 10 月 2 日

— 誤差のいろいろ —

1. 丸め誤差

全ての数値は「コンピュータで扱える数値」に近似される。

例 1　$e^{\pi\sqrt{163}}$ は、本当は無理数

262537412640768743.99999999999925007259719818568887 9...

なのだが、有効数字を 30 桁以上に設定しないと

262537412640768744.00000000000

という整数値が出力される。

```
fpprec : 100$
y : exp(bfloat(%pi) * sqrt(bfloat(163)))$
printf(false, "precision = ~d", fpprec);
printf(false, "exp(pi * sqrt(163)) = ~h", y);
```

2. オーバーフロー・アンダーフロー

大き過ぎる数は仕様の限界を超えて overflow のエラーを起こす。小さ過ぎる数は正しく表現できなかったり 0 になってしまったりする。

例 2　$(1/10^n) \times 10^n = \ ?$

```
#include<stdio.h>
main()
{
    int i, n;
    float x;
    for(n = 35; n <= 50; n++){
        x = 1.0;
        for(i = 1; i <= n; i++) x /= 10;
        for(i = 1; i <= n; i++) x *= 10;
        printf("n = %d のとき %f\n",n,x);
    }
}
```

```
n = 38 のとき 1.000000
n = 39 のとき 1.000000
n = 40 のとき 0.999995
n = 41 のとき 0.999967
n = 42 のとき 1.000527
n = 43 のとき 0.994922
n = 44 のとき 0.980909
n = 45 のとき 1.401299
n = 46 のとき 0.000000
n = 47 のとき 0.000000
```

3. 累積誤差

塵も積もれば山となる。

例 3 $\displaystyle\sum_{i=1}^{n} \sin\left(\frac{2\pi i}{n}\right)$ の理論値は 0 だが ...

```
#include<stdio.h>
#include<math.h>
#define pi 3.14159265358979324
main()
{
    int i,j,n;
    float x;
    for(j=1;j<=7;j++){
        n=(int)pow(10,j);
        x=0.0;
        for(i=1;i<=n;i++) x+=sin((2*pi*i)/n);
        printf("n = %8d のとき %9.6f\n",n,x);
    }
}
```

```
n =       10 のとき -0.000000
n =      100 のとき  0.000001
n =     1000 のとき -0.000015
n =    10000 のとき -0.000050
n =   100000 のとき -0.000020
n =  1000000 のとき -0.028050
n = 10000000 のとき  0.078272
```

4. 積み残し

$|a| \gg |b|$ のときに $a+b$ を計算すると b の下の方の桁が無視される。

例 4 $\dfrac{1}{1^2}+\dfrac{1}{2^2}+\dfrac{1}{3^2}+\cdots$ の理論値は $\dfrac{\pi^2}{6} = 1.6449340668\cdots$ だが ...

```
#include<stdio.h>
#define pi 3.14159265358979324
main()
{
    int i,j,n;
    float x;
    printf("理論値は %f だが\n\n",pi*pi/6);
    printf("昇順に加えると\n",n);
    for(j = 2; j <= 7; j++){
        n = (int)pow(10,j);
        x = 0.0;
        for(i = 1; i <= n; i++) x += 1.0/i/i;
        printf("%8d 項目までの和 = %f\n",n,x);
    }
    printf("\n");
    printf("降順に加えるとまして\n",n);
    for(j = 2; j <= 7; j++){
```

```
        n = (int)pow(10,j);
        x = 0.0;
        for(i = n; i >= 1; i--) x += 1.0/i/i;
        printf("%8d 項目までの和 = %f\n",n,x);
    }
}
```

理論値は 1.644934 だが

昇順に加えると

```
     100 項目までの和 = 1.634984
    1000 項目までの和 = 1.643935
   10000 項目までの和 = 1.644725
  100000 項目までの和 = 1.644725
 1000000 項目までの和 = 1.644725
10000000 項目までの和 = 1.644725
```

降順に加えるとましで

```
     100 項目までの和 = 1.634984
    1000 項目までの和 = 1.643934
   10000 項目までの和 = 1.644834
  100000 項目までの和 = 1.644924
 1000000 項目までの和 = 1.644933
10000000 項目までの和 = 1.644934
```

例 5 $\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots=+\infty$ だが ...

```
#include<stdio.h>
#include<math.h>
main()
{
    int c = 0, i = 0;
    float x = 0.0, y = -1.0;
    while(x != y){
        i++; c++;
        y = x;
        x += 1.0/i;
        if(c == 100000){
            printf("1/1 + 1/2 + ... + 1/%7d = %9.6f\n",i,x);
            c = 0;
        }
    }
    printf("ここから先は和の値が増えなくなりました:\n");
    printf("1/1 + 1/2 + ... + 1/%7d = %9.6f\n",i,x);
}
```

```
1/1 + 1/2 + ... + 1/ 100000 = 12.090851
1/1 + 1/2 + ... + 1/ 200000 = 12.782757
1/1 + 1/2 + ... + 1/ 300000 = 13.195325
(中略)
1/1 + 1/2 + ... + 1/1800000 = 15.120297
1/1 + 1/2 + ... + 1/1900000 = 15.215665
1/1 + 1/2 + ... + 1/2000000 = 15.311032
ここから先は和の値が増えなくなりました:
1/1 + 1/2 + ... + 1/2097152 = 15.403683
```

5. 桁落ち

$a \quad b$ のときに $a - b$ を計算すると著しく精度が落ちる。

例 6 $\left(\dfrac{1}{n} - \dfrac{1}{n+1}\right) \times n \times (n+1) = 1$ のはずが ...

```
#include<stdio.h>
main()
{
    int n, i;
    float x, y, z, w;
    n = 10;
    for(i = 0; i < 8; i++){
        x = 1.0 / n;
        y = 1.0 / (n + 1);
        z = x - y;
        w = z *  (n * (n + 1.0));
        printf("n = %9d,   [1/n-1/(n+1)]*(n*(n+1)) = %f\n", n, w);
        n *= 10;
   }
}
```

```
n =        10,   [1/n-1/(n+1)]*(n*(n+1)) = 1.000000
n =       100,   [1/n-1/(n+1)]*(n*(n+1)) = 0.999999
n =      1000,   [1/n-1/(n+1)]*(n*(n+1)) = 1.000075
n =     10000,   [1/n-1/(n+1)]*(n*(n+1)) = 0.999817
n =    100000,   [1/n-1/(n+1)]*(n*(n+1)) = 1.000454
n =   1000000,   [1/n-1/(n+1)]*(n*(n+1)) = 1.023183
n =  10000000,   [1/n-1/(n+1)]*(n*(n+1)) = 1.421086
n = 100000000,   [1/n-1/(n+1)]*(n*(n+1)) = 0.000000
```

6. おまけその１：どこがバグかな？

$\frac{1}{1}+\frac{1}{2}+\frac{1}{3}+\cdots$ の計算をしているはずなのに

```
#include<stdio.h>
main()
{
    int n;
    float x = 0.0;
    for(n = 1; n < 11; n++){
        x += 1 / n;
        if(n == 1) printf("1/1                    = %9.6f\n", x);
        if(n == 2) printf("1/1 + 1/2              = %9.6f\n", x);
        if(n == 3) printf("1/1 + 1/2 + 1/3        = %9.6f\n", x);
        if(n >  3) printf("1/1 + 1/2 + ... + 1/%2d = %9.6f\n", n, x);
    }
}
```

```
1/1                    =  1.000000
1/1 + 1/2              =  1.000000
1/1 + 1/2 + 1/3        =  1.000000
1/1 + 1/2 + ... + 1/ 4 =  1.000000
1/1 + 1/2 + ... + 1/ 5 =  1.000000
1/1 + 1/2 + ... + 1/ 6 =  1.000000
1/1 + 1/2 + ... + 1/ 7 =  1.000000
1/1 + 1/2 + ... + 1/ 8 =  1.000000
1/1 + 1/2 + ... + 1/ 9 =  1.000000
1/1 + 1/2 + ... + 1/10 =  1.000000
```

7. おまけその２：おもしろい配列の仕様

```
/* array.c
おもしろい配列の仕様
理由は a[i] = *(a+i) = *(i+a) = i[a] だからなんだけど、わかるかな？
*/

#include<stdio.h>
main()
{
    int  a[] = {1,4,2,8,5,7}, i;
    char b[] = "Kochi";

    for(i = 0; i < 6; i++) printf("%d[a] = %d, ", i, i[a]);
    printf("\n");
    for(i = 0; i < 5; i++) printf("%d[b] = %c, ", i, i[b]);
}
```

```
 0[a] = 1,  1[a] = 4,  2[a] = 2,  3[a] = 8,  4[a] = 5,  5[a] = 7,
 0[b] = K,  1[b] = o,  2[b] = c,  3[b] = h,  4[b] = i,
```