数値計算の基礎Ｉ

０．目次

１．情報落ち

計算のルールを「10進4桁、切り捨て」と仮定する。
２つの数の加算では、まず小数点が合わされ、大きい数が優先される。
したがって、12.34 + 0.005678 は 12.34 と計算される。このように、
絶対値の小さい数を絶対値の大きい数に加えてもほとんど影響を
与えない現象を情報落ちという。

２．オーバーフロー・アンダーフロー

計算結果の絶対値がコンピュータの処理できる最大の数を越えてしまう
現象をオーバーフローという。

計算結果の絶対値がコンピュータの処理できる最小数以下になる現象を
いう。アンダーフローが生じると計算結果を０にすることが多い。

３．桁落ち

12.34 － 12.33 = 0.01 のように、同符号で絶対値のほぼ等しい２個の
数に対して、減算を行うと有効数字が著しく減ってしまう現象をいう。

４．丸め誤差の累積

コンピュータの内部で処理できる数値の桁数が決まっているため、
丸め（４捨５入、切り捨て、切り上げ）が起こる。このために生じる
誤差を丸め誤差という。

１．情報落ち

計算のルールを「10進4桁、切り捨て」と仮定する。２つの数の加算では、まず小数点が合わされ、大きい数が優先される。したがって、12.34 + 0.005678 は 12.34 と計算される。

	1	2	.	3	4				
＋		0	.	0	0	5	6	7	8
	1	2	.	3	4				

このように、絶対値の小さい数を絶対値の大きい数に加えてもほとんど影響を与えない現象を**情報落ち**という。

［**例１**］10**nに1を加え情報落ちが発生したら表示する。

●プログラム(m111.c)

```
1  /* << m111.c >> */
2  #include <stdio.h>
3
4  int main() {
5    int n;
6    float a,b;
7    double da,db;
8
9    /* 単精度(float)の場合。*/
10   printf("単精度(float) ¥n");
11   a = 1.0e5;
12   for( n=6; n<=9; n++ ) {
13     a = a*10.0;
14     b = a + 1.0;
15     printf("10**%d + 1 = %f ",n,b);
16     if(a == b) { printf("情報落ち発生"); }
17     printf("¥n");
18   }
19
20   /* 倍精度(double)の場合。*/
21   printf("倍精度(double) ¥n");
22   da = 1.0e14;
23   for( n=15; n<=17; n++ ) {
24     da = da*10.0;
25     db = da + 1.0;
26     printf("10**%d + 1 = %lf ",n,db);
27     if(da == db) { printf("情報落ち発生"); }
28     printf("¥n");
29   }
30 }
```

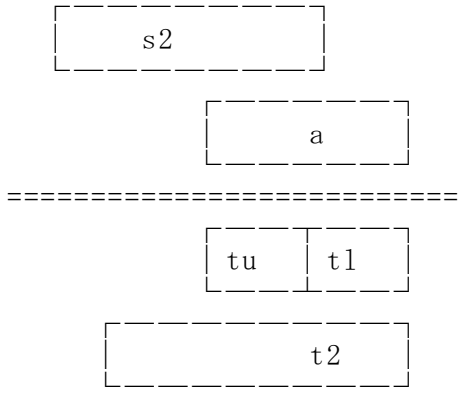
実行結果

```
% cc m111.c
% ./a.out
単精度(float)
10**6 + 1 = 1000001.000000
10**7 + 1 = 10000001.000000
10**8 + 1 = 100000000.000000 情報落ち発生
10**9 + 1 = 1000000000.000000 情報落ち発生
倍精度(double)
10**15 + 1 = 1000000000000001.000000
10**16 + 1 = 10000000000000000.000000 情報落ち発生
10**17 + 1 = 100000000000000000.000000 情報落ち発生
```

［**例２**］「定数aをn回加える」操作を３つの方法で比較する。

方法１：順にaをn回加える。情報落ちが発生する。

方法２：順にaをn回加えるが、情報落ちをできるだけ防ぐ工夫をする。



tlは情報落ちのため加算に反映されない部分

すなわち、aをtuとtlに分割し、tuはs2に加え、tlはt2に加える。
このようにすると、情報落ちで捨てられるtlがt2に保存され、
情報落ちをある程度防ぐことができる。

方法３：精度を上げて、順にaをn回加える。

●プログラム(m121.c)

```
/* << m121.c >> */
#include <stdio.h>

int main() {
  int i,n;
  float a,s1,s2,t1,t2,tl,tu,w;
  double s3;

  while( 1 ) {
    /* aとnの読み込み。*/
    scanf("%f%d",&a,&n);
    if( (a==0)||(n==0) ) { break; }
    printf("a=%10.8f  n=%8d  n*a=%16.8f ¥n",a,n,n*a);

    /* 方法１：順に加えていく方法。情報落ちが生じる。*/
```

```
16      s1 = 0.0;
17      for( i=1; i<=n; i++ ) {
18        s1 = s1 + a;
19      }
20      printf("方法１ : %16.8f ¥n",s1);
21
22      /* 方法２ */
23      s2 = 0.0; t2 = 0.0;
24      for( i=1; i<=n; i++ ) {
25        w = s2 + a;
26        tu = w - s2;
27        t1 = a - tu;
28        t2 = t2 + t1;
29        s2 = s2 + tu;
30      }
31      printf("方法２ : %16.8f ¥n",s2+t2);
32
33      /* 方法３ : 倍精度で計算。*/
34      s3 = 0.0;
35      for( i=1; i<=n; i++ ) {
36        s3 = s3 + a;
37      }
38      printf("方法３ : %16.8lf ¥n",s3);
39    }
40  }
```

実行結果

```
1  % cc m121.c
2  % ./a.out
3  1.001 1000
4  a=1.00100005  n=     1000  n*a=    1001.00006104
5  方法１ :    1000.99121094
6  方法２ :    1001.00006104
7  方法３ :    1001.00004673
8  1.001 10000
9  a=1.00100005  n=    10000  n*a=   10010.00000000
10 方法１ :   10009.78027344
11 方法２ :   10010.00000000
12 方法３ :   10010.00046730
13 1.001 100000
14 a=1.00100005  n=   100000  n*a= 100100.00781250
15 方法１ :  100047.93750000
16 方法２ :  100099.99218750
17 方法３ :  100100.00467300
18 0.0 0
```

２．オーバーフロー・アンダーフロー

計算結果の絶対値がコンピュータの処理できる最大の数を越えてしまう現象を**オーバーフロー**という。

［**例１**］オーバーフローの確認。

●プログラム(m211.c)

```
1  /* << m211.c >> */
2  /* 10**nを計算していき、オーバーフローを調べる。*/
3  #include <stdio.h>
4
5  int main() {
6    int n;
7    float f;
8    double d;
9
10   /* 単精度の場合。*/
11   printf("単精度(float) ¥n");
12   f = 1.0e+36;
13   for( n=37; n<=40; n++ ) {
14     f = f * 10.0;
15     printf("[%2d] %e ¥n",n,f);
16   }
17
18   /* 倍精度の場合。*/
19   printf("倍精度(double) ¥n");
20   d = 1.0e+306;
21   for( n=307; n<=310; n++ ) {
22     d = d * 10.0;
23     printf("[%3d] %le ¥n",n,d);
24   }
25 }
```

実行結果

```
1  % m211.c
2  % ./a.out
3  単精度(float)
4  [37] 1.000000e+37
5  [38] 1.000000e+38
6  [39] Inf              オーバーフロー発生
7  [40] Inf
8  倍精度(double)
9  [307] 1.000000e+307
10 [308] 1.000000e+308
11 [309] Inf              オーバーフロー発生
12 [310] Inf
```

計算結果の絶対値がコンピュータの処理できる最小数以下になる現象をいう。
アンダーフローが生じると計算結果を０にすることが多い。

［例２］アンダーフローの確認。

●プログラム(m221.c)

```
/* << m221.c >> */
/* 10**nを計算していき、アンダーフローを調べる。*/
#include <stdio.h>

int main() {
  int n;
  float f;
  double d;

  /* 単精度の場合。*/
  printf("単精度(float) ¥n");
  f = 1.0e-42;
  for( n=43; n<=47; n++ ) {
    f = f / 10.0;
    printf("[%2ld] %e ¥n",n,f);
  }

  /* 倍精度の場合。*/
  printf("倍精度(double) ¥n");
  d = 1.0e-316;
  for( n=317; n<=325; n++ ) {
    d = d / 10.0;
    printf("[%3ld] %le ¥n",n,d);
  }
}
```

実行結果

```
% cc m221.c
% ./a.out
単精度(float)
[43] 9.949219e-44
[44] 9.809089e-45
[45] 1.401298e-45
[46] 0.000000e+00        アンダーフロー発生
[47] 0.000000e+00
倍精度(double)
[317] 9.999997e-318
[318] 9.999987e-319
[319] 9.999889e-320
[320] 9.999889e-321
[321] 9.980126e-322
[322] 9.881313e-323
[323] 9.881313e-324
[324] 0.000000e+00        アンダーフロー発生
[325] 0.000000e+00
```

［例３］アンダーフロー・オーバーフローの回避。

ａのｎ乗(ａ：実数、ｎ：整数)をできるだけ正確に計算する問題を考察する。

2.0 の 1000 乗 ≒ 0.107151×10の302 乗
2.0 の -1000 乗 ≒ 0.933264×10の-301 乗

となるが、aのｎ乗を計算する場合、工夫をしなければ、オーバーフロー・アンダーフローが発生する。

数値を仮数部（0.1以上1未満）と指数部に分けて表現し、乗算の結果、仮数部が1以上または0.1未満になると、0.1以上1未満になるように調整することで、オーバーフロー・アンダーフローを回避できる。

○5の2乗の場合。

初期値。
1.0 ＊ 10 の 0 乗
5を掛ける。
（調整前）5.0 ＊ 10 の 0 乗
（調整後）0.5 ＊ 10 の 1 乗
5を掛ける。
（調整前）2.5 ＊ 10 の 1 乗
（調整後）0.25 ＊ 10 の 2 乗

5の(-2)乗の場合。
まず、5の2乗を求め、その後逆数を計算する。
5の2乗：0.25 ＊ 10 の 2 乗
5の(-2)乗 = (1/0.25) ＊ 10 の (-2)乗
= 4.0 ＊ 10 の (-2)乗
= 0.4 ＊ 10 の (-1)乗

●プログラム(m231.c)

```
 1  /* << m231.c >> */
 2  #include <stdio.h>
 3
 4  int main() {
 5    int e,i,m,n;
 6    double a,f,w;
 7
 8    while( 1 ) {
 9      /* aとnの読み込み。*/
10      scanf("%lf%d",&a,&n);
11      if( a == 0 ) { break; }
12
13      m = n; if( n < 0 ) { m = -n; }
14      f = 1.0; /* f：仮数部。*/
15      e = 0;   /* e：指数部。*/
16      for( i=1; i<=m; i++ ) {
17        f = f * a;
18        while( f >= 1 ) {
19          f = f/10.0; e++;
20        }
21        while( f < 0.1 ) {
22          f = f*10.0; e--;
23        }
24      }
25      if( n < 0 ) {
26        f = 1.0/f;
27        e = -e
28      }
29      while( f >= 1 ) { f = f/10.0; e++; }
30      while( f < 0.1 ) { f = f*10.0; e--; }
31      printf("%lf の %d 乗 = %lf×10の%d 乗 ¥n",a,n,f,e);
32    }
33  }
```

実行結果

```
% cc m231.c
% ./a.out
2.0 1000
2.000000 の 1000 乗 = 0.107151×10の302 乗
2.0 -1000
2.000000 の -1000 乗 = 0.933264×10の-301乗
0.0 0
```

３．桁落ち

12.34 - 12.33 = 0.01 のように、同符号で絶対値のほぼ等しい２個の数に対して、減算を行うと有効数字が著しく減ってしまう現象をいう。

［**例１**］桁落ちの確認。

２元連立１次方程式

$$ax + by = e$$
$$cx + dy = f$$

を a=0.780, b=0.563, c=0.913, d=0.659, e=0.217, f=0.254 で解くと、

$$x = \frac{de - bf}{ad - bc} \qquad y = \frac{af - ce}{ad - bc}$$

厳密解は、x=1.0, y=-1.0 となる。数値解は、x=1.03125, y=-1.043295 となる。この場合、桁落ちが de-bf、ad-bc、af-ce で起こっている。

●プログラム(m311.c)

```
 1  /* << m311.c >> */
 2  #include <stdio.h>
 3
 4  int main() {
 5    float a,b,c,d,e,f,x,y;
 6
 7    a = 0.780; b = 0.563; c = 0.913; d = 0.659;
 8    e = 0.217; f = 0.254;
 9    printf("d*e    :%f ¥n",d*e);
10    printf("b*f    :%f ¥n",b*f);
11    printf("d*e-b*f:%f ¥n",d*e-b*f);
12    printf("a*d    :%f ¥n",a*d);
13    printf("b*c    :%f ¥n",b*c);
14    printf("a*d-b*c:%f ¥n",a*d-b*c);
15    printf("¥n");
16    x = (d*e - b*f)/(a*d - b*c);
17    y = (e - a*x)/b;
18    printf("x = %f ¥n",x);
19    printf("y = %f ¥n",y);
20  }
```

実行結果

```
 1  % cc m311.c
 2  % ./a.out
 3  d*e    :0.143003
 4  b*f    :0.143002
 5  d*e-b*f:0.000001
 6  a*d    :0.514020
 7  b*c    :0.514019
 8  a*d-b*c:0.000001
 9
10  x = 1.031250
11  y = -1.043295
```

［例２］桁落ちの発生と回避。

円周率（π = 3.14159 26535 89793 …）を半径1の円を内部から近似する正2^n角形の周囲の長さで求めることを考える。

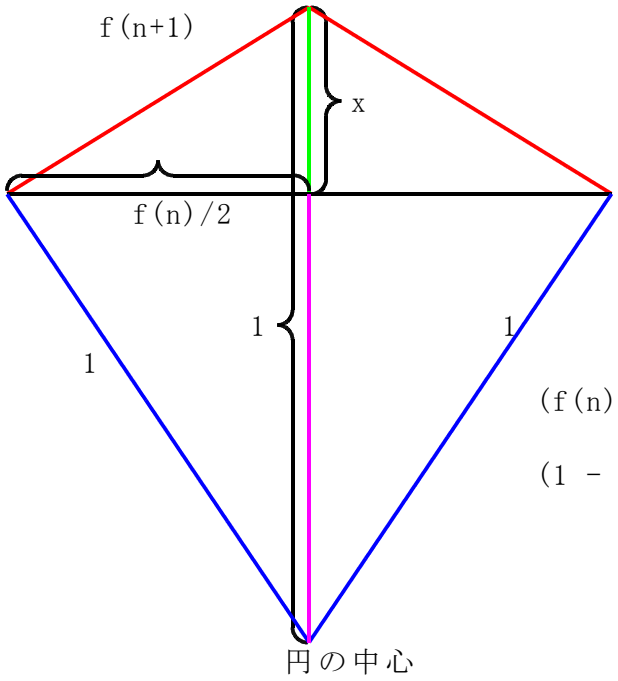
最初、正 2^2 角形で近似し、一辺の長さをf(2)= $\sqrt{2}$ とする。

正2^n角形の一辺f(n)と正2^{n+1}角形の一辺f(n+1)の関係は、

$$f(n+1) = \sqrt{2 - \sqrt{4 - f(n)^2}}$$

である。円周率は、$\frac{2^n f(n)}{2}$ となる。

（ヒント）



$(f(n)/2)^2 + x^2 = f(n+1)^2$

$(1 - x)^2 + (f(n)/2)^2 = 1$

円の中心

●プログラム(m321a.c)

桁落ちが発生する。

```
1  /* << m321a.c >> */
2  #include <stdio.h>
3  #include <math.h>
4
5  int main() {
6    int n;
7    double f,pi,w;
8
9    pi = 4.0*atan(1.0);
```

```
10    printf("π  = %16.12lf ¥n",pi);
11    printf("  n             f(n)         π-近似値¥n");
12
13    w = 4.0; /* w=2^2。 */
14    f = sqrt(2.0);
15    for( n=2; n<=31; n++ ) {
16      f = sqrt(2.0 - sqrt(4.0 - f*f));
17      w = w * 2.0; /* w=2^n。 */
18      printf("%2d: ",n);
19      printf("%15.12f %15.12f¥n",f,pi-w*f/2);
20    }
21  }
```

実行結果

```
% cc m321a.c -lm
% ./a.out
π  =   3.141592653590
  n             f(n)         π-近似値
 2:  0.765366864730  0.080125194669
 3:  0.390180644032  0.020147501332
 4:  0.196034280659  0.005044163044
 5:  0.098135348655  0.001261496635
 6:  0.049082457046  0.000315402657
 7:  0.024543076571  0.000078852446
 8:  0.012271769298  0.000019713222
 9:  0.006135913526  0.000004928310
10:  0.003067960373  0.000001232085
11:  0.001533980638  0.000000307979
12:  0.000766990375  0.000000077045
13:  0.000383495195  0.000000020127
14:  0.000191747599 -0.000000001218
15:  0.000095873799  0.000000008269
16:  0.000047936899  0.000000046214
17:  0.000023968452 -0.000000257350
18:  0.000011984231 -0.000001471605
19:  0.000005992120 -0.000003900115
20:  0.000002996060 -0.000003900115
21:  0.000001498067 -0.000081611432
22:  0.000000749071 -0.000237028299
23:  0.000000374609 -0.000858618904
24:  0.000000187305 -0.000858618904
25:  0.000000094243 -0.020685006579
26:  0.000000047122 -0.020685006579
27:  0.000000025810 -0.322508961548
28:  0.000000014901 -0.858407346410
29:  0.000000000000  3.141592653590
30:  0.000000000000  3.141592653590
31:  0.000000000000  3.141592653590
```

（考察）有理化

$$\frac{a-b}{\sqrt{a}-\sqrt{b}} = \frac{a-b}{\sqrt{a}-\sqrt{b}}\,\frac{\sqrt{a}+\sqrt{b}}{\sqrt{a}+\sqrt{b}} = \sqrt{a}+\sqrt{b}$$

有理化によって、分母の桁落ちを防ぐことができる。

この場合、f(n)は0に近づくので、f(n)からf(n+1)を計算するときに、桁落ちが生じている。桁落ちを防ぐために、f(n)の右辺を有理化する。

g(2) =	$\sqrt{2}$
g(n+1)=	$\dfrac{g(n)}{\sqrt{2+\sqrt{4-g(n)^2}}}$

円周率は、2^n*g(n)/2　となる。

●プログラム(m321b.c)

桁落ちを回避する。

```
1  /* << m321b.c >> */
2  #include <stdio.h>
3  #include <math.h>
4
5  int main() {
6    int n;
7    double g,pi,w;
8
9    pi = 4.0*atan(1.0);
10   printf("π = %16.12lf ¥n",pi);
11   printf("  n             g(n)          π-近似値¥n");
12
13   w = 4.0; /* w=2^2。 */
14   g = sqrt(2.0);
15   for( n=2; n<=31; n++ ) {
16     g = g / sqrt(2.0 + sqrt(4.0 - g*g));
17     w = w * 2.0; /* w=2^n。 */
18     printf("%2d: ",n);
19     printf("%15.12f %15.12f¥n",g,pi-w*g/2);
20   }
21 }
```

実行結果

```
% cc m321b.c -lm
% ./a.out
π =   3.141592653590
  n            g(n)        π-近似値
 2:  0.765366864730  0.080125194669
 3:  0.390180644032  0.020147501332
 4:  0.196034280659  0.005044163044
 5:  0.098135348655  0.001261496635
 6:  0.049082457046  0.000315402657
 7:  0.024543076571  0.000078852445
 8:  0.012271769298  0.000019713223
 9:  0.006135913526  0.000004928313
10:  0.003067960373  0.000001232079
11:  0.001533980637  0.000000308020
12:  0.000766990375  0.000000077005
13:  0.000383495195  0.000000019251
14:  0.000191747598  0.000000004813
15:  0.000095873799  0.000000001203
16:  0.000047936900  0.000000000301
17:  0.000023968450  0.000000000075
18:  0.000011984225  0.000000000019
19:  0.000005992112  0.000000000005
20:  0.000002996056  0.000000000001
21:  0.000001498028  0.000000000000
22:  0.000000749014  0.000000000000
23:  0.000000374507  0.000000000000
24:  0.000000187254  0.000000000000
25:  0.000000093627 -0.000000000000
26:  0.000000046813 -0.000000000000
27:  0.000000023407 -0.000000000000
28:  0.000000011703 -0.000000000000
29:  0.000000005852 -0.000000000000
30:  0.000000002926 -0.000000000000
31:  0.000000001463 -0.000000000000
```

４．丸め誤差の累積

コンピュータの内部で処理できる数値の桁数が決まっているため、丸め（４捨５入、切り捨て、切り上げ）が起こる。このために生じる誤差を**丸め誤差**という。丸め誤差が累積して異常な結果となる例を示す。

［**例**］つぎの漸化式で決まる数列を考える。

$f(n) = f(n-1) + f(n-2) \quad (n>=2),\ f(1)=b,\ f(0)=1$

$b = \frac{1-\sqrt{5}}{2} = -0.618034...$

厳密解は、$f(n)=b^n$ となる。

数値解をg(n)とし、g(1)にbを代入したときに生じた丸め誤差をeとすると、丸め誤差eの累積していく様子はつぎのようになる。

```
g(0) = f(0)       = 1
g(1) = f(1)+e     = b+e
g(2) = g(1)+g(0) = f(1)+e+f(0)      = f(2)+e
g(3) = g(2)+g(1) = f(2)+e+f(1)+e    = f(3)+2e
g(4) = g(3)+g(2) = f(3)+2e+f(2)+e   = f(4)+3e
g(5) = g(4)+g(3) = f(4)+3e+f(3)+2e = f(5)+5e
g(6) = g(5)+g(4) = f(5)+5e+f(4)+3e = f(6)+8e
g(7) = g(6)+g(5) = f(6)+8e+f(5)+5e = f(7)+13e
...
```

●プログラム(m411.c)

```
/* << m411.c >> */
#include <stdio.h>
#include <math.h>

int main() {
  int i,n;
  float b,f[999],g[999];

  n = 100; b = (1.0 - sqrt(5.0))/2.0;
  printf("               厳密解          数値解 ¥n");
  f[0] = 1.0; f[1] = b;
  g[0] = 1.0; g[1] = b;
  printf("[  0] %15.6e %15.6e ¥n",f[0],g[0]);
  printf("[  1] %15.6e %15.6e ¥n",f[1],g[1]);

  for( i=2; i<=n; i++ ) {
    f[i] = b * f[i-1];
    g[i] = g[i-1] + g[i-2];
    if( i%10 == 0 ) {
      printf("[%3d] %15.6e %15.6e ¥n",i,f[i],g[i]);
    }
  }
}
```

実行結果

```
% cc m411.c -lm
% a.out
                厳密解          数値解
[  0]    1.000000e+00    1.000000e+00
[  1]   -6.180340e-01   -6.180340e-01
[ 10]    8.130620e-03    8.129716e-03
[ 20]    6.610700e-05   -4.494190e-05
[ 30]    5.374910e-07   -1.365757e-02
[ 40]    4.370134e-09   -1.679836e+00
[ 50]    3.553191e-11   -2.066062e+02
[ 60]    2.888964e-13   -2.541088e+04
[ 70]    2.348907e-15   -3.125332e+06
[ 80]    1.909807e-17   -3.843904e+08
[ 90]    1.552792e-19   -4.727690e+10
[100]    1.262516e-21   -5.814675e+12
```

(考察)

厳密解f(n)と数値解g(n)の違いが顕著になる理由は、丸め誤差eにかかる係数が、フィボナッチ数のように大きくなるからである。