誤差の課題

グループ名

大きな数どおしのわずかな差

1.

大きな数どおしのわずかな差は，丸め誤差にとくに影響を受ける. 23.173-23.094 を有効数字がそれぞれ5桁，4桁，3桁，2桁で計算した結果を示せ. 同様に，0.81321/(23.173-23.094) を有効数字がそれぞれ5桁，4桁，3桁，2桁で計算した結果を示せ．

In [1]:
```
from decimal import *

def pretty_p(result,a,b,operator):
    print('context.prec:{}'.format(getcontext().prec))
    print(' %20.14f' % (a))
    print( '%1s%20.14f' % (operator, b))
    print('-----------')
    print( ' %20.14f' % (result))

n = 5
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
```

```
context.prec:5
   23.17300000000000
-  23.09400000000000
-----------
    0.07900000000000
```

In [2]:
```
n = 4
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
```

```
context.prec:4
   23.17000000000000
-  23.09000000000000
-----------
    0.08000000000000
```

In [3]:
```
n = 3
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
```

```
context.prec:3
   23.20000000000000
-  23.10000000000000
-----------
    0.10000000000000
```

In [4]:
```
n = 2
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
```

```
context.prec:2
   23.00000000000000
-  23.00000000000000
-----------
    0.00000000000000
```

In [5]:
```
n = 5
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
```

```
context.prec:5
   23.17300000000000
-  23.09400000000000
-----------
    0.07900000000000
```

2

In [6]:
```
n = 5
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
c=Decimal('0.81321').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
c/(a-b)
```

```
context.prec:5
   23.17300000000000
-  23.09400000000000
-----------
    0.07900000000000
```

Out[6]: Decimal('10.291')

In [7]:
```
n = 4
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
c=Decimal('0.81321').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
c/(a-b)
```

```
context.prec:4
   23.17000000000000
-  23.09000000000000
```

```
-----------
    0.08000000000000
```

Out[7]: Decimal('10.12')

In [8]:
```
n = 3
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
c=Decimal('0.81321').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
c/(a-b)
```

```
context.prec:3
   23.20000000000000
-  23.10000000000000
-----------
    0.10000000000000
```

Out[8]: Decimal('8')

In [9]:
```
n = 2
getcontext().prec = n

a=Decimal('23.173').quantize(Decimal(10) ** -(n-2))
b=Decimal('23.094').quantize(Decimal(10) ** -(n-2))
c=Decimal('0.81321').quantize(Decimal(10) ** -(n-2))
pretty_p(a-b,a,b,'-')
c/(a-b)
```

```
context.prec:2
   23.00000000000000
-  23.00000000000000
-----------
    0.00000000000000
```

```
---------------------------------------------------------------------------
DivisionByZero                            Traceback (most recent call last)
<ipython-input-9-0b4e940db466> in <module>
      6 c=Decimal('0.81321').quantize(Decimal(10) ** -(n-2))
      7 pretty_p(a-b,a,b,'-')
----> 8 c/(a-b)

DivisionByZero: [<class 'decimal.DivisionByZero'>]
```

複利計算

10 進数10桁および3桁の有効桁数をもった計算機になったつもりで，以下の条件で預金を求める計算をおこなえ．

1. 元本を10000万円とする
2. 利息0.3%とする
3. 複利計算で10年でいくらになるか．

In [10]:
```
from decimal import *

n = 10
digits = Decimal(10) ** -n
getcontext().prec = n

x = Decimal('0.1').quantize(digits)*10000*10000
print(x)
rate = Decimal('0.3').quantize(digits)/100
print(rate)
x*(1+rate)**10
```

```
10000000.00
0.0030000000
```

Out[10]: Decimal('10304082.57')

In [11]:
```
n = 3
digits = Decimal(10) ** -n
getcontext().prec = n

x = Decimal('0.1').quantize(digits)*10000*10000
print(x)
rate = Decimal('0.3').quantize(digits)/100
print(rate)
x*(1+rate)**10
```

```
1.00E+7
0.003
```

Out[11]: Decimal('1.00E+7')

2次方程式解の公式の罠

係数を a = 1, b = 10000000($= 10^7$), c = 1としたときに, 通常の解の公式を使った解と, 解と係数の関係(下記の記述を参照）を使った解とを出力するプログラムをpythonで作成すると以下の通りとなる．解の有効数字が2種類の計算方法の違いでどう違うか，いくつかの精度で実行させた結果を使って解説せよ．

In [6]:
```
from numpy import sqrt
def solve_normal_formula(a,b,c):
    x0=(-b-sqrt(b**2-4*a*c))/(2*a)
    x1=(-b+sqrt(b**2-4*a*c))/(2*a)
    return (x0,x1)

def solve_precise_formula(a,b,c):
    x0=(-b-sqrt(b**2-4*a*c))/(2*a)
    x1=c/(a*x0)
    return (x0,x1)

b = 10**7

print(solve_normal_formula(1,b,1))
print(solve_precise_formula(1,b,1))
```

```
(-9999999.9999999, -9.96515154838562e-08)
(-9999999.9999999, -1.00000000000001e-07)
```

In [3]:
```
from decimal import *

prec = 4 #14
b= '10000000'
print("\n b =", b)
print("\nprec=", prec)
getcontext().prec = prec
print(solve_normal_formula(Decimal('1'),
                  Decimal(b),
```

```
                    Decimal('1')))
print(solve_precise_formula(Decimal('1'),
                    Decimal(b),
                    Decimal('1')))
```

```
b = 10000000

prec= 4
(Decimal('-1.000E+7'), Decimal('0E+4'))
(Decimal('-1.000E+7'), Decimal('-1E-7'))
```

下の例のdecimalでprecを指定したのがわかりやすいが，精度が高く求められる計算法(solve_precise_formula)では，低い精度でも正しい解が求められている．一方，解の公式を用いた計算法(solve_normal_formula)では，すぐに引き算のせいで０という間違った値になってしまう．

これを通常のdouble floatのnumpyへ戻して考えてみると，bの大きさによって，10^7では３桁目で誤差が現れている程度に求まっているが，10^8では間違った答え０となってしまう．