Chapter ９. 実数と計算

９－１．実数の精度

９－１－１．実数の誤差と限界

たとえば、1/3.0 * 3.0は、通常は1.0になります。3.0で割る操作を100回繰り返して、もう一度100回3.0倍する操作を繰り返しても、理論的には同じことで結果は1.0になるはずです。それでは実際に次のようなプログラムの断片の部分を実行させてみましょう。

```
double x = 1.0;
for (int i=0; i<100; i++ ) { x=x/3.0; }
for (int i=0; i<100; i++ ) { x= x*3.0; }
System.out.println( "result is "+x );
```

ところが、表示は次のようになりました。1.0にはなっていません。どうしてでしょうか？

```
result is 1.0000000000000009
```

Javaでは内部的には、有限のデジタル情報として実数を表しています。そのために、本来アナログ情報である実数を表し切れない場合があるのです。本来のアナログ情報である実数とデジタルで表した場合の実数との差を誤差と呼んでいます。実数の計算には、誤差がつきものなのです。Javaでは、実数を64bitの高精度で表していますので、従来のプログラミング言語で頻繁に起きていた誤差がかなり少なくなっています。しかし、上のように単純な割り算・掛け算も繰り返して行なわせると誤差が入り込んできます。

９－１－２．丸め誤差と打ち切り誤差

以上で表わしたようなJavaの実数の計算の中入り込んでくる誤差を丸め誤差（Round　Error）と呼んでいます。誤差の多かった時代には、なるべく丸め誤差を少なくするような形で実数の計算が行なわれていました。誤差が発生しやすいときはどんなときでしょうか？経験的に、次のようなことがよく言われています。

「違うオーダー（大きさ）の数を足し算したり、引き算したりすると誤差が発生する」

たとえば、非常に大きな数と小さな数を足し合わせることを考えてみてください。100,000,000に1を足すときは、100,000,000であっても100,000,001であっても、数の大きさ的には大きくは変わりません。同じように、有限のデジタル情報として実数が表されるとき、小さな差は丸められてしまいます。この丸めが誤差を発生する原因となっています。実数で計算を行なう場合は、このような大きさが著しく違う数を足したり、引いたりすることを避けなければなりません。

また、無理数の√２や、πなどを考えてみてください。Javaでは、有限のビット数でしか実数を表わせませんので、どこかで桁を打ち切るしかありません。このようにして入り込んでくる誤差を打ち切り誤差（Drop Error）と呼んでいます。以下のJavaのプログラムの断片は、それらを表示するものです。詳しい内容は、次の節で説明します。ここで表示された範囲だけしか、計算に入れていません。それ以下の小数部は無視されています。

```
System.out.println(  Math.PI );
System.out.println(  Math.sqrt( 2.0 ) );
```

MacintoshのJDK 1.4.2の環境では、以下のように表示されました。

１０－２．　実数関係のクラスライブラリィ

１０－２－１．Mathクラスについて

Mathクラスには、数学上使う便利な関数がメソッドとして用意されています。「もう数学なんて嫌！」っていう人が多いかも知れません。しかし、いざというとき（ないと思っていると案外とあるでしょう）には必要となってきます。特に、コンピュータグラフィックスのときには、必ず使われます。

メソッド	評価結果の型	意味
E	double	自然対数の底を表す
PI	double	円周率を表す
ceil(実数)	double	大きいか等しい一番小さな整数に丸める
floor(実数)	double	小さいか等しい一番大きな整数に丸める
rint(実数)	double	一番近い整数に丸める
round(実数)	int, long	実数を整数に四捨五入して丸める
acos(実数)	double	$\cos^{-1}$を求める（$-\pi/2 \sim \pi/2$の間の角度）
asin(実数)	double	$\sin^{-1}$を求める（$-\pi/2 \sim \pi/2$の間の角度）
atan(実数)	double	$\tan^{-1}$を求める（$-\pi/2 \sim \pi/2$の間の角度）
atan2(実数 , 実数)	double	２つの実数をx、y座標とする極角度を求める
cos(角度)	double	cosを求める
sin(角度)	double	sinを求める
tan(角度)	double	tanを求める
abs(数)	double, float, long, int	絶対値を返してくる
exp(実数)	double	自然対数のべき乗を求める
log(実数)	double	自然対数を求める
pow(実数, 実数)	double	べき乗を求める
random( )	double	0.0以上1.0未満の間で乱数を返してくる
sqrt(実数)	double	平方根を求める

実際にこれらの関数を使って簡単な数式、あるいは代入文を記述してみましょう。

★整数と実数との間の変換に使われる関数

```
double  ceil = Math.ceil( 44.23 );          // ceil = 45.0
double  floor = Math.floor( -44.23 );       // floor = -45.0
double  closest =  Math.rint( -44.23 );     // closest = - 44.0
int     round = Math.round( -44.23 );       // round = - 44
```

整数への変換で注意しなければならないのは、ceilやfloorなどは、それぞれ負の数は、絶対値的には異なるような数に変換されるということです。また、roundだけは整数に変換されますが、他のものは関数の計算結果が実数のままであるということにも注意してください。また、roundは四捨五入ですので、下の例のような形になります。コメントとして書かれているのが、計算結果です。

```
Math.ceil( 23.45 )      // 24.0 Math.ceil( - 23.45 )      //  - 23.0
Math.floor( 23.45 )     //  23.0      Math.floor( - 23.45 )     //  - 24.0
Math.round( 23.45 )     // 23         Math.round( - 23.45 )     //  - 23
Math.round( 23.50 )     // 24         Math.round( -23.50 )      //  -23
Math.round( 23.55 )     // 24         Math.round( -23.55 )      //  -24
```

```
(int) 23.45          // 23          (int) 23.45          // - 23
```

ceilとfloorの関係、あるいはroundとfloorの関係を示すのに次のような公式が用いられることがあります。これで、各関数の関係を判断してください。

```
Math.ceil( x ) = - Math.floor( - x )
Math.round( x ) = (int) Math.floor( x + 0.5 )
```

★三角関数、逆三角関数

三角関数、逆三角関数の場合は、角度を指定するときはすべてradian体系（πを基準とする体系）で行なわれます。通常の３６０度を使って角度を表す体系はdegree体系と呼ばれています。その間の変換は、次のようになります。なお、πを指定するときは、JavaではMath.PIと記述します。

```
radian角度 = degree角度 / 180 * π                    例えば 90° は1/2π
```

これを変換するために、Java2からは、以下の２つのメソッドが用意されました。

```
Math.toRadians( degree角度 )          // radian角度に変換する
Math.toDegrees( radian角度 )          // degree角度に変換する
```

実際に、三角関数などを使った式を記述してみましょう。

```
double alpha = Math.acos( 0.33 );                    // cos-1 0.33
double alpha = Math.atan2( 0.0, -2.33 );        // alpha = pi
double mycos = Math.cos( Math.PI / 3 );              // cos 60
```

★その他の関数

その他の関数としては、絶対値を求めるようなもの、自然対数関連のもの、べき乗や平方根などがあります。絶対値以外は、実数に対してしか用意されていないことに注意してください。

```
int    abs = Math.abs( -44.23 );          // 44.23
double ep = Math.exp( Math.PI );          // e^π
double naturallog = Math.log( 1.34e5 );   // log_e 1.34e5
double power = Math.pow( 4, 3 );          // 4^3
double rand = Math.random( );             // 0.0 <= n < 1.0
double root = Math.sqrt( 2.0 );           // 1.414....
```

Math.randomは、乱数になっていて、0.0〜1.0の間の適当な実数を発生して返してくれます。しかし、1.0という値は発生しませんので注意が必要です。たとえば、サイコロのように1から6までのどれかの整数を得たい場合は、次のように記述します。

```
int dice = (int)(Math.floor( Math.random( ) * 6 ) + 1);          // 1〜6までの乱数を発生
```

実は、これは乱数としてはあまり出来がよくありません。もう少しばらつかせたい場合は、かなり大きな数を掛けて、余りを取るという方式が良く採られます。

```
int betterdice = (int)(Math.random( ) * 534) % 6 + 1;          // 534は適当な大きな数
```

なお、Java2以降は、このようなことをしなくても、だいぶ乱数のばらつきが良くなりました。改善されたようです。

９－３．等比数列と金利計算

社会系の学問でも実数を使った様々な計算が行なわれますが、ここでは一番身近な金利の計算を実数を用いて行なってみましょう。

９－３－１．等差数列と等比数列

次の節で出てくる複利計算のように、一定の率で、掛け算されていくような数列を等比数列と呼びます。等差数列と比べてみます。初項をa、公差をd、公比をrとすると、それぞれ次のようになります。

等差数列の第ｎ項　　$a_n = a + (n -1) * d;$

等比数列の第ｎ項　　$a_n = a * r^{(n - 1)}$

例をあげてみましょう。たとえば、a = 1、 dおよびrを両方とも2と置くと次のようになります。

等差数列	1	3	5	7	9	11	13	15
等比数列	1	2	4	8	16	32	64	128

数列にはいろいろな公式がありますが、等差数列も等比数列も、for文などの繰返しを使えば、直感的に求めることができます。たとえば、次のプログラムの断片は、第１０項までの等差数列の総和を求めています。

```
int     total = 0;
int     a = 1, d = 2;
for ( int i = 1; i <=10 ; i++ ) {
        total  =  total + a;
        System.out.print(   "第" + i  + "項は "  +   a   +  "      "  )
        a = a + d;
}
System.out.println( "" );
System.out.println( "ここまでの総和は" + total );
```

なお、等差数列も等比数列も第ｎ項までの総和を求める公式があります。等比数列は、次の金利計算のところで紹介します。等差数列は、以下のようになります。

total = n * (2 * a + (n - 1) * d　　) / 2　　初項と公差から

total = n * (a + z) / 2　　初項と末項から（ z = a_n ）

９－３－２．金利計算の基礎

★単利法と複利法

定期預金やローン計算などの利息を計算するときには、複利法を使います。元金に利率を掛けた結果が利息になりますが、単利法で利息を計算する場合と複利法で利息を計算する場合は、期間が複数に渡る場合には利息の額が異なってきます。

単利法…元金に期間数と利率を掛けたものが利息になります
複利法…期間毎に元金と利息の合計に利率が掛かかったものが次の利息になります

それでは、例えば定期預金で10,000円を3年預けた場合を考えてみましょう。年利率1%とします。３年後に額はいくらになるでしょうか？

単利法…10000 + 10000 * 0.01 * 3 → 10,300円

複利法…(10000 * (1 + 0.01)) * (1 + 0.01) * (1 + 0.01) → 10,303円

単利法と複利法では以上のように差がでてきます。利息の計算は通常は複利法で計算します。複利法の場合、貯める場合はより利息が大きくなります。逆にお金を借りる場合は、最初の利息が高くなります。また、数学的には複利法での計算は、等比数列の計算の応用となっています。

★年率、月率、日率

通常利息を使われるの年利率（年率あるいは年利）は、１年における利率を示しています。しかし、月で返済するようなローンや、日で返済するようなローンなどの場合は、１月の利率（月率）、あるいは１日の利率（日率）を求める必要があります。さらに、複利を使った定期預金でも、半年複利のものもありますし、ローンでもボーナス払い（年に２回と考えます）を別に用意しているものもあります。その場合には、半年の利率が利息計算に使われることになります。

半年の利率	→	年率を2で割ればよい
月率	→	年率を12で割ればよい（注１）
日率	→	年率365(366)で割ればよい

「金利」という言葉は通常は年率を示しています。注意しなければならないのは、消費者金融などで使われている実質年率と名目年率（約定年率）の違いです。

名目年率…預金金利や貸出金利のように契約で定められている年利率を指します。
実質年率…名目年率から物価上昇率（インフレ率）を差し引いた年利率です。

たとえば、名目年率が30%でも、その年の物価上昇率が5%であった場合には実質年率は25%になります。法律的には、年率は40%を上限としています。消費者金融の実質年率が28%と表記されていましても、名目年率はそれよりも数%高いのです。実生活でも注意したいものです。

★固定金利と変動金利

利率が一定の場合は、固定金利と呼びます。それに対して、変化するものを変動金利と呼びます。変動金利は、１年毎に利率が見直されるものと半年毎に利率が見直されるものとがあります。

固定金利	年利率が一定
変動金利	年利率が相場に応じて変わる
上限つき変動金利	変動金利だが、上限（これ以上高くならない限度）はある

★定期預金の場合

それでは、定期預金を想定して、10000円を預けて、固定金利で、年利率が5.5%の場合（注２）、10年でいくらになるか、毎年表示してみましょう。毎年結果として求まる額を終価と呼びます。変数baseには元金が、変数totalには終価が代入されます。また変数rateは、年利率が代入されています。totalとrateは、実数型の変数になっていることに注意してください。

```
public class Financial {
    public static void main( String [] arg ) {
        int    base = 10000;                    // 元金
        double total = base,  rate  = 0.055;          // 終価と年利率

        for ( int year = 1; year <= 10; year ++ ) {
            total  =  total * (1+rate);
            System.out.println( year + "年後は、" + total + "円" );
        }
    }
```

```
}
```

なお、等比数列の公式を用いれば、繰返しを使わなくとも、次の式で10年後の終価を計算することができます。Math.powerの引数は両方とも実数なので、10年を10.0と記述しています。

```
double total =  base * Math.pow(  1+rate,  10.0 ) ;
```

※注１　正確には、月率は各月で日数が違うので日率をその月の日数分合計したものになります。ここで12で割っているのは概算で求める場合、あるいは月率を一定にする場合に使われる計算方法です。
※注２　こんなに高い預金金利は不況下の日本では外国債ぐらいしかありません。

９－３－３．年金計算

ここで言う「年金」とは老後のための積立金のことではありません。財務的には広く定期的にお金を積み立てたり、払ったりするもののことを年金と呼ぶのです。貯める場合と返済する場合で年金の呼び方が違います。

貯める場合の年金…積立金
返済する場合の年金…賦金

★期末払いと期首払い

積立金では、期首払い（それぞれの期の最初で年金を払う）場合と期末払い（それぞれの期の最後で年金を払う）場合では、利息が微妙に異なってきます。

期首払い…年金にも利息がつく
期末払い…年金には利息が付かない

たとえば、1万円の元金があって、年利率0.2%で毎年1000円の積立金であるような預金の場合に、期首払いでは１年後は11,022円ですが、期末払いでは11,020円となります。2円の差は、期首払いの場合に年金にも利息が付いたからです。賦金の場合は、期末払いがほとんどですが、その期に借りている額に対して利息が発生します。

★定期積立預金の場合

元金として10,000円を預けて、固定金利で年利率が5.5%の場合、毎年5000円を預けていくと、10年でどれくらいになるか終価を表示してみましょう。積立金は期末払いとします。

```
public class Saving {
        public static void main( String [ ] arg ) {
                int       base = 10000, deposit = 5000;        // 元金と積立金
                double total = base,   rate  = 0.055;                  // 終価と年利率

                for ( int year = 1; year <= 10; year ++ ) {
                        total  =  total * (1+rate) + deposit;
                        System.out.println(  year + "年後は、" + total + "円" );
                }
        }
}
```

なお、等比数列の和の公式を用いれば、次の式で10年後の終価を求めることができます。

```
double total =  base * Math.pow(  1+rate,  10.0 ) +
                  deposit * (Math.pow( 1+rate, 10.0 ) - 1) / rate;
```

９－３－４．賦金（ふきん）計算

借入金を返済する方式には次のような３つの方式があります（注１）。それぞれの場合に、返済のイメージが異なります。

元金均等方式	月々決まった額の元金に利息を加えた金額を返済する方式
元利均等方式	借入元金と借入利息を合わせて月々、同じ金額を返済する方式
元金分割元金均等方式	借入元金が大きい場合、元金を何年かごとに分割して、元利均等方式で返済する方式

★元利均等返済の場合

月々の返済額は一定になります。しかし最初の方の返済は、利息の部分の比率が非常に高くて、利息分を返済していることになります。なかなか、元金（借入金）は小さくなりません。返済がある程度進むと元金分の割合も増えて利息は小さくなっていきますが、消費者金融などの年率の高いローンでは、月々の返済額を少なくすると利息ばかりを返済することになり、元金がほとんど返せなくなってしまうでしょう。返済総額は他の方式よりも大きくなります。多くのローンはこの返済方法を採用しています。

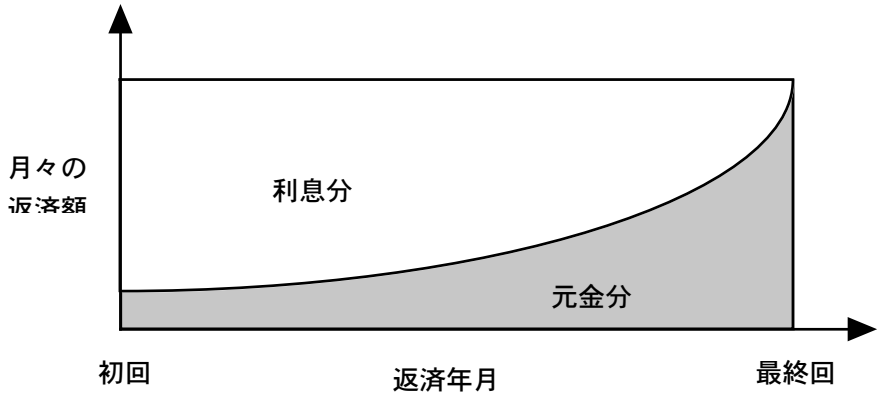

図9-1　元利均等返済

★元金均等返済の場合

元金と併せて、利息分も返済するので、最初の方の返済では月々の返済金額が非常に大きくなります。しかし、元金を毎回の返済で払っていきますので、利息は一定の比率で減少していきます。そのため、利息の総額としては元利均等返済よりもだいぶ少なくなります。

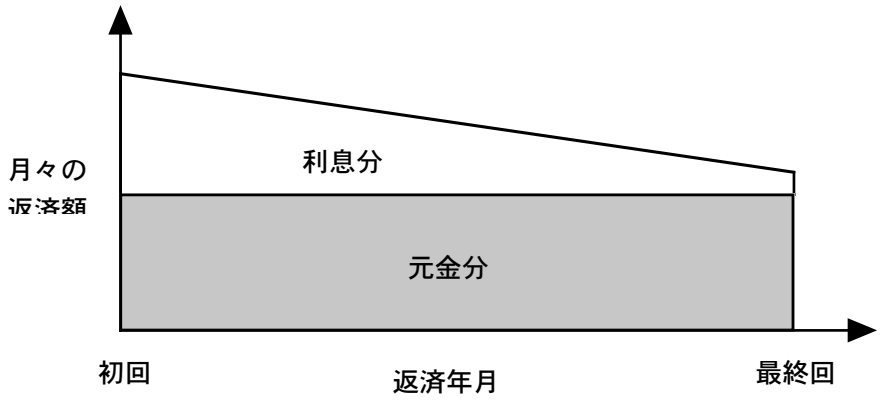

図9-2　元金均等返済

★元金分割（ステージ式）元金均等返済

借入金が大きい場合、元金均等返済では最初の方の回の支払い金額が膨大になってしまいます。これを避けるために、借入金をいくつかのステージに分割して、各ステージごとで、元金均等返済を行なう方式です。借入金の規模が大きい中小企業への融資などに良く使われるとのことです。

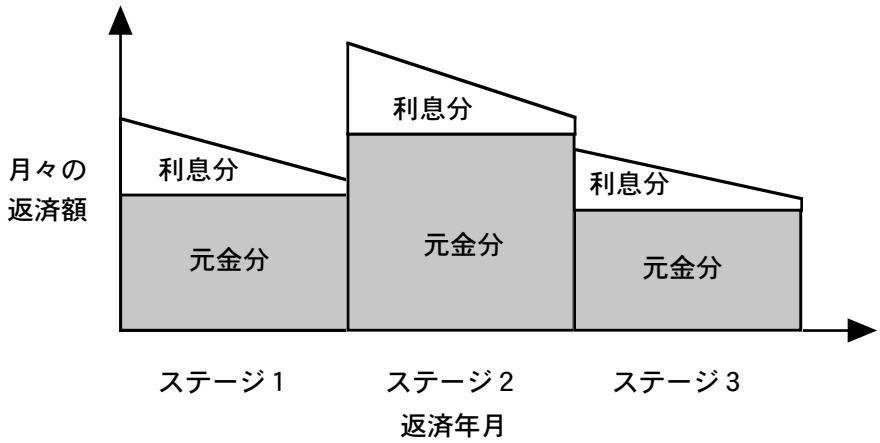

図9-3　元金分割元金均等返済

こんなこと関係ないと思っている方も多いでしょう。しかし、プログラミングばかり目を向けていて実際の社会の金利システムを勉強をしていないと、思わぬところで落とし穴に落ちることになります。

★ローン返済の場合（何カ月掛かるか？）

実際に賦金（ローン）の返済をプログラミングしてみましょう。たとえば、無謀にも300万円のBMWを買ってしまったとします。頭金が100万円として、固定金利で年利率が14.5%のときに、元利均等返済で月々50000円ずつ期末払いで払っていくと、何カ月で返せるか計算してみましょう。支払った月数と、残金を表示していきます。totalが毎回の返済残高、rateは12で割っていますので月率を表し、loanが月々の支払額を示しています。

```
public class Installment {
        public static void main( String [] arg ) {
                int     left = 3000000-1000000, loan = 50000;       // 借入金と月々の支払額
                double total = left,  rate  = 0.145/12;       // 支払い後の残金と月の利率

                for ( int month = 1; total > 0 ; month ++ ) {
                        total  =  total * (1+rate) - loan;
                        System.out.println(  month + "月後の残金、" + total + "円" );
                }
        }
}
```

★ローン返済の場合（月々いくら必要か？）

300万円のBMWを買おうと計画しました。幸い？頭金は50万円はあるとします。残りを5年ローン（つまり60回払い）で買いたいときに、元利均等返済で、利率は固定金利・複利で年利率14.5%で、期末払いという条件で、一体月々幾ら払えば良いのでしょう？外側の繰返しで、月々払うお金を10000円から始めて、100円単位で上げていき、払いきれるかどうか試行しています。内側の繰返しでは60回分払ってみています。60回払った後の残高を表示していきます。もし、60回後に残高が0円よりも小さくなっていたら、月々その金額で払えば払い切れることになります。

```
public class Loan {
        public static void main( String [ ] arg ) {
                int     base = 3000000-500000;
```

```
                    double total,  rate  = 0.145/12;

                    for ( int  loan = 10000 ; ; loan += 100 ) {
                            total = base;
                            for  ( int  month = 1; month <= 60; month ++ ) {
                                    total  =  total * (1+rate) - loan;
                            }
                            System.out.println( "月々の支払:" + loan +
                                                    " 60回後の残金: "+ total  );
                            if  ( total  <  0 ) { break; }
                    }
            }
    }
```

月々の支払額を求めるには等比数列の和の公式を使えば、上のような繰返しを使わなくとも、等比数列の公式から、次のような式で求めることができます。また、利息の総額を支払総額から元金を引くことにより求めることができます。

```
double  loan = base *  Math.pow( 1+rate,  60.0 ) * rate / ( Math.pow( 1+rate, 60.0 ) - 1 );
double  interest = loan * 60 -  base;
```

９－５．課題

課題9-1.
元金100,000円で、月々5,000円を入れる定期預金を開設して、ちょっと幸福になりました。しかし、100万円貯まるのは一体いつでしょう？元利均等返済の期末払いで、固定複利の年利率0.3%として、何カ月後に100万円を越すか計算するプログラムを作りなさい。クラス名は、Million。何カ月後になるか解答も求めなさい。

課題9-2.
JaguarF-Typeという1,023万円のオープンカーを買うことにしました。無謀にも手持ちは0円です。元利均等返済の期末払いで、固定複利の年利率14.5%として、7年ローンとして、月々幾らか払えば良いか計算するプログラムを作りなさい。1円単位まで計算しなさい。クラス名は、Jaguarやっぱりでしょ。解答も求めなさい。

図9-4 Jaguar F-Type

ヒント：15万円ぐらいから、払いきれるかどうか試し始めるとよいでしょう。