実験番号 _____

実験題目　　計算機における計算誤差の解析

実験年月日

R2 年 5 月 ２０ 日 5 時限 ～ 7 時限

天候 _____ 気温 _____[℃] 湿度_____[%]

R2 年 5 月 ２６ 日 5 時限 ～ 7 時限

天候 _____ 気温 _____[℃] 湿度_____[%]

R2 年 6 月 2 日 5 時限 ～ 7 時限

天候 _____ 気温 _____[℃] 湿度_____[%]

実験レポート提出者

電子情報工学科 第 3 学年 8 番

氏名　　織田 祐斗

提出年月日 R2 年 6 月 _____日 提出

１．目的

ある入力データを計算機によって計算処理を施したとき，結果として得られる値と真の値との差を誤差という。

計算機は計算を絶対に間違えないという考え方がある。確かに計算機は正常な条件下では計算を正確に繰り返す。しかし，計算機には本来，丸め誤差や桁落ちなどと呼ばれる誤差があり，計算するたびに毎回同じ誤差が発生し，計算過程を通して誤差の伝搬が正確に繰り返される。したがって計算の内容によっては，誤差の累積によって真の値から著しくはずれた思わぬ結果を得ることがある。

ここで，計算機による数値計算特有の誤差を解析してみる。

２．課題１

N= 120 に対する N の階乗を求めるプログラムによって整数型および倍精度実数型のそれぞれの演算結果を出力し，計算機内部での数値表現の精度について考察せよ。

２．１　仕様

k:　整数、階乗の計算結果を格納
d:　倍精度実数、階乗の計算結果を格納
i:　整数、カウンタ

２．２　段階的詳細化

●i←1
●k←1
●d←1.0
●”　　整数型　　実数型”を表示
■i≦20 である限り
｜●k←k×i
｜●d←d×i
｜●”i!=　　k　　d”を表示
｜●i←i＋1
■

２．３　プログラムリスト

```
#include<stdio.h>

int main(void){
        int i=1,k=1;
        double d=1.0;
        printf("                整数型                実数型¥n");
        while(i<=20){
                k=k*i;
                d=d*i;
                printf("%2d!=%22d%22.1lf¥n",i,k,d);
                i++;
        }
        return 0;
}
```

２．４　結果

	整数型	実数型
1!=	1	1.0
2!=	2	2.0
3!=	6	6.0
4!=	24	24.0
5!=	120	120.0
6!=	720	720.0
7!=	5040	5040.0
8!=	40320	40320.0
9!=	362880	362880.0
10!=	3628800	3628800.0
11!=	39916800	39916800.0
12!=	479001600	479001600.0
13!=	1932053504	6227020800.0
14!=	1278945280	87178291200.0
15!=	2004310016	1307674368000.0
16!=	2004189184	20922789888000.0
17!=	-288522240	355687428096000.0
18!=	-898433024	6402373705728000.0
19!=	109641728	121645100408832000.0
20!=	-2102132736	2432902008176640000.0

13!以降から整数型と倍精度実数型で計算結果に差が生じた

２．５　考察

計算していくと、倍精度実数型は正しく計算が行えており、整数型は桁数が足りなくなっていることから、整数型では計算中に桁数の限界を超えた値を格納しようとしたために正確な値を格納できなかったと考えられる

３．課題２

1⊕εM>1 にもとづくプログラムによって倍精度の計算機イプシロン"M を求め，使用機種の浮動小数点体系について考察せよ。

３．１　仕様

d:　倍精度実数、計算結果を格納
i:　倍精度実数
k;　整数、桁数

３．２　段階的詳細化

●i←0.5
●d← 1 .0
●k←1
■1+i＞1 である限り
｜●k←k＋ 1
｜●d←d÷ 2
｜●i←d÷ 2
■
●“倍精度実数での桁数は k、浮動小数点数は d です”を表示

３．３　プログラムリスト

```
#include<stdio.h>
int keta(void);

int main(void){
        int k=1;
        double i=0.5,d=1.0;
        while(1+i>1){
                k++;
                d=d/2;
                i=d/2;
        }
        printf("倍精度実数での桁数は%d、浮動小数点数は%e です¥n",k,d);

        return 0;
}
```

３．４　結果

倍精度実数での桁数は 53、浮動小数点数は 2.220446e-16 です

３．５　考察

計算機イプシロンは、その計算機内で１より大きい最小の値である。得られた値から、使用機器では倍精度実数型を５３桁の２進数で表現しているようである

４．課題３

自然対数の底を求める問題について，

s1＝1+1/1!+1/2!+……+1/n!、

s2=1/n!+1/(n-1)!+1/(n-2)!+……+1/2!+1/1!+1、

s3=((……((1/n+1)1/(n-1)+1)1/(n-2)+……+1)1/2+1)1/1+1

の三通りの方法で計算するプログラムを作成し，それぞれの計算結果を比較・ 検討してみよ。

４．１　仕様

a:　倍精度実数、計算結果を格納
b:　倍精度実数、計算結果を格納
c;　倍精度実数、計算結果を格納
n;　整数
k:　倍精度実数
i:　整数、カウンタ
j:　 倍精度実数

４．２　段階的詳細化

●n を入力
●i← 1
●k← 1
●a← 1 .0
■i≦n である限り
｜●k←k×i
｜●a←a+(1÷k)
｜●i←i+ 1
■
●i←n
●b←1÷k
■i＞ 0 である限り
｜●k←k÷i
｜●j←(1÷k)
｜●b←b+j
｜●i←i-1
■
●c←1÷n

●n←n－1
■n＞0 である限り
｜●j←c+1
｜●c←j×(1÷n)
｜●n←n－1
■
●c←c+1
●“方法１：a”を表示
●“方法２：b”を表示
●“方法３：c”を表示

４．３　プログラムリスト

```
#include<stdio.h>

int main(void){
	int n,i=1;
	double a=1.0,b,c,k=1,j;
	printf("n＝");
	scanf("%d",&n);
	while(i<=n){
		k=k*i;
		a=a+(1/k);
		i++;
	}
	i=n;
	b=1.0/k;
	while(i>0){
		k=k/i;
		j=1.0/k;
		b=b+j;
		i=i-1;
	}
	c=1/(double)n;
	n=n-1.0;
	while(n>0){
		j=c+1.0;
```

```
                c=j*(1.0/(double)n);
                n=n-1.0;
        }
        c=c+1.0;
        printf("方法１：%20.19e¥n",a);
        printf("方法２：%20.19e¥n",b);
        printf("方法３：%20.19e¥n",c);
        return 0;
}
```

４．４　結果

```
n＝10
方法１：2.718281801146384513e+00
方法２：2.718281801146384513e+00
方法３：2.718281801146384513e+00
n＝20
方法１：2.718281828459045534e+00
方法２：2.718281828459045090e+00
方法３：2.718281828459045090e+00
n＝30
方法１：2.718281828459045534e+00
方法２：2.718281828459045534e+00
方法３：2.718281828459045090e+00
```

４．５　考察

方法１＞方法２＞方法３の順に計算誤差が生じやすくなっている(n=30 の方法３の値が正しいと仮定した場合)。計算方法では、方法２は方法１と比べて、小さな値から計算しており、方法３は方法２と比べて、できる限り小数点以下の桁数が小さくなる値で計算している。このことから、方法１と方法２は丸め処理を行う際に生じる誤差の違い(前者は比較的大きな値から、後者は比較的小さな値から丸め処理を行っている)、方法２と方法３は丸め処理を行う前の小数点以下の桁数の差の違い(前者は桁数が多く、後者は少ない)が、計算結果に影響を与えたと考えられる

５．課題４

二次方程式

ax^2+bx+c= 0

を解くためのプログラムを作成し，

x=(-b±√D)/2a、D=b^2-4ac

を使う場合と

x=2c/(-b±√D) 、D=b^2-4ac

を使う場合とを比較・検討してみよ。ただし，A＝1.0，C＝1.0 とする。B は 10.0，100.0，1000.0，10000.0，100000.0 と係数の値を変えていくこと。

５．１　仕様

main 関数

a: 倍精度実数、定数

b; 倍精度実数、定数

c: 倍精度実数、定数

x1: 倍精度実数、求める値

x2: 倍精度実数、求める値

d: 倍精度実数、判別式の値

s: 倍精度実数

sqrt 関数

x: 倍精度実数

s: 倍精度実数

n: 倍精度実数

５．２　段階的詳細化

main 関数

●a を入力

●b を入力

●c を入力

●d←b×b-4×a×c

●s←sqrt(d)

●x1←((b×-1)+s)÷(2×a)

●x2←((b×-1)-s)÷(2×a)

▲d≠０

｜●”方法１：x1、x2”を表示

＋――――――――――――――――

｜●”方法１：x”を表示
▼
●x1←(2×c)÷((b×-1)+s)
●x2←(2×c)÷((b×-1)-s)
▲d≠０
｜●”方法２：x1、x2”を表示
＋――――――――――――――――
｜●”方法２：x”を表示
▼

sqrt 関数
●x を代入
●s←x÷2
●n←0
■s が n でない限り
｜●n←s
｜●s←(s+x÷s)÷2
■
●s の値を返す

５．３　プログラムリスト

```
#include<stdio.h>
#include<math.h>

int main(void){
        double a,b,c,d,s,x1,x2;
        printf("a=");
        scanf("%lf",&a);
        printf("b=");
        scanf("%lf",&b);
        printf("c=");
        scanf("%lf",&c);
        d=b*b-(4*a*c);
        s=sqrt(d);
        x1=(((-1)*b)+s)/(2*a);
        x2=(((-1)*b)-s)/(2*a);
```

```
	if(d!=0){
		printf("方法１：%lf,%lf¥n",x1,x2);
	}else{
		printf("方法１：%lf¥n",x1);
	}
	x1=(2*c)/(((-1)*b)+s);
	x2=(2*c)/(((-1)*b)-s);
	if(d!=0){
		printf("方法２：%lf,%lf¥n",x1,x2);
	}else{
		printf("方法２：%lf¥n",x1);
	}
	return 0;
}

double sqrt(double x){
	double s=x/2.0,n=0.0;
	while(s!=n){
		n=s;
		s=(s+x/s)/2.0;
	}
	return s;
}
```

５．４　結果

```
a=1.0
b=10.0
c=1.0
方法１：-0.101021,-9.898979
方法２：-9.898979,-0.101021
a=1.0
b=100.0
c=1.0
方法１：-0.010001,-99.989999
方法２：-99.989999,-0.010001
a=1.0
b=1000.0
```

c=1.0
方法１：-0.001000,-999.999000
方法２：-999.999000,-0.001000
a=1.0
b=10000.0
c=1.0
方法１：-0.000100,-9999.999900
方法２：-9999.999889,-0.000100
a=1.0
b=100000.0
c=1.0
方法１：-0.000010,-99999.999990
方法２：-99999.966146,-0.000010

５．５　考察

b が 10000.0、100000.0 に、小さい値の計算結果に誤差が生じている。
方法１では、b が 4ac より非常に大きいと D が b^2 に近づいてしまうために、分子の値が０に近い数になるので、計算結果の誤差が方法２よりも大きくなってしまう。
そのため、計算誤差が大きくなりやすい方法１よりも、方法２の方がより正確な計算結果を導き出せることになる。

６．感想、意見

今回は同一のコードで、違う結果が求められることが多く、修正に苦労した。
もうマシンの気分に振り回されたくないと感じた。