# 数値計算講義　第2 回
# 級数の和・ 打ち切り 誤差

金子　晃

kanenko@mbk.nifty.com
http://kanenko.a.la9.jp/

# 前回の復習



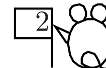
コンピュータと純粋数学は**丸め誤差**のために事態が異なる

- コンピュータの通常変数はサイズが限られている．
  - 整数なら，16 ビット，32 ビット ... 桁数が限られる．
    それを越えるとオーバーフローのエラーとなる．
  - 実数なら，単精度，倍精度 ... 有効桁が限られた有限小数．
    それを越えるとオーバーフロー・アンダーフローのエラーとなる．

  数学では，連続的に $h \to 0$ とできるが，
  計算機では，$h = \varepsilon$ (計算機イプシロン) の次はいきなり 0 に変わる．

- 小数点のずれた二つの実数の和では**情報落ち**が生ずる．
  - 三つ以上の和が順序に依存する．
  - 引き算により**桁落ち**も生ずる．(結合法則が成り立たない)
- コンピュータの内部ではすべてのデータは 0 と 1 で表示される．
  - 整数：二進整数．
  - 小数：IEEE754 規格．
  - 文字：文字コード．(ieee754.c に Kerosoft Ltd. を食べさせた人は要猛反省!)

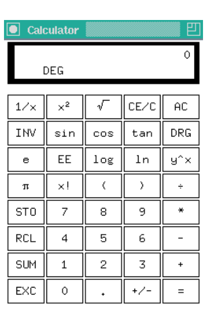
# コンピュータでどうやって計算する？




電卓:

ほとんどの OS には電卓が備わっている!

左: WindowsXP の電卓アクセサリ
(表示メニューで関数電卓も選べる)

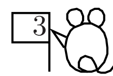



右: X Window の xcalc

マウスでボタンを押すのは使いにくい. 本物の電卓の方がまし.
(ちょっとした計算だけなら, 電卓を探す手間が省けるという程度.)

これらの関数電卓エミュレータの精度はどれくらいなんだろう？
C 言語の double で計算しているんじゃない?
本物の関数電卓はせいぜい 10 桁表示だが, それにしては計算が正確だ???

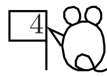
コンピュータで計算するにはどうすればよいか？




- **インタープリタ:**
入力に対して, すぐに答を返してくれるようなソフト
(コマンドライン版電卓)
例: BASIC インタープリタ, Python, OCaML, CLISP, Dr Scheme, R など

- **コンパイラ:**
エディターでソースプログラムを書き, それを機械語に直して実行する.
実行するまでの準備が長いが, 実行は高速.
例: FORTRAN, Pascal, C など

- **代数計算や数式処理のソフト:**
インタープリタの拡張版としても使える.
例: Maxima, Risa/Asir, Pari/GP など




例題として, $\displaystyle\sum_{i=1}^{N} \frac{1}{i^2}$ を計算してみよう.

$N$ が, 例えば 10 と決まっていれば, インタープリタに

$$1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \frac{1}{5^2} + \frac{1}{6^2} + \frac{1}{7^2} + \frac{1}{8^2} + \frac{1}{9^2} + \frac{1}{10^2}$$

と書けばよい.
$N = 100$ じゃ書く気しない. (^^;

シグマ記号が使えたら良いな.
実は, 使えるプログラミング言語がもう有る.
でも実際にコンピュータがやる仕事は同じ.

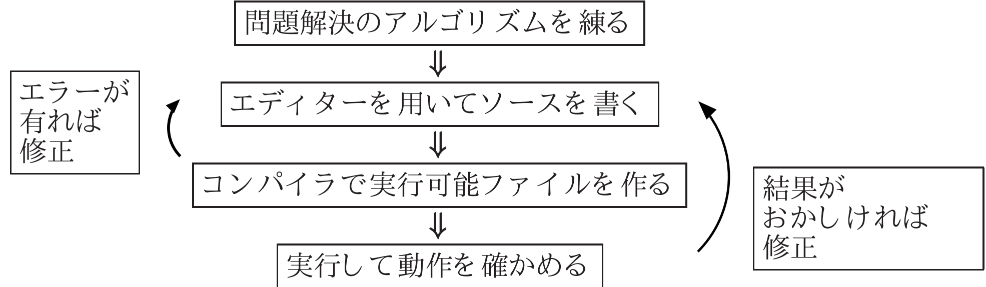
疑似コード

```
s  <-  0                      和を入れる変数を初期化
for i:=1 to N do              ループを作る
begin
    s  <-  s + 1/i^2           第 i 項を加える
end
print s                       結果を出力
```

# 実際の言語による実装 (implement) のプロセス



※ **疑似コード**とは，アルゴリズムを分かりやすく表現するための
疑似プログラミング説明文のこと．
これに使われる言語を疑似言語という．
理論計算機の世界では，Pascal 言語的なものが良く使われる．




※ 上に戻る矢印の工程を**デバッグ**という．
**デバッガー**はこの過程を助けるツール (gdb など).
※ 実は3 番目の四角の中は, コンパイラとリンカに分かれる．

プログラミングの技術は本質的ではない.
本質はアルゴリズムで, プログラミングはそれを書き下すだけ．

# 実際の言語による実現例 1 — C 言語

C のコード num2-1.c

```
#include<stdio.h>
#include<stdlib.h>
int main(void)
{
    int i,N;
    double s;

    printf("Give N : ");
    scanf("%d",&N);
    s=(double)0;
    for (i=1;i<=N;i++)
    {
        s=s+(double)1/i/i;
    }
    printf("%22.15lf\n",s);
    return 0;
}
```

- (double)1/i/i のところを 1/i/i と書くと，答が 1 になってしまう．
  1.0/i/i と書いても同じようにみえるが，処理系によっては，一旦
  単精度で計算したものを倍精度に変換して s に加えるので危険．
- (double)1/i/i と (double)1/(i*i) も同じとは限らない！
  後者では，先に $i*i$ を整数として計算する処理系が多く，
  前者よりも早めにオーバーフローが起こり得る．

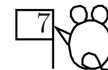
**コンパイルの仕方**：上のソースファイルを num2-1.c という名前で作ったら，

```
gcc num2-1.c -o num2-1
```

で実行ファイル num2-1 ができる．

# 実際の言語による実現例 2 — Pascal 言語





Pascal のコード num2-1.p

```
program zeta2;              (* ソースファイル num2-1.p *)
var i,N: integer;           (* 整数型変数の宣言 *)
var s: double;              (* 倍精度実変数の宣言 *)
begin
    write('Give N :');
    readln(N);              (* 加える項の総数を入力 *)
    s:=0;                   (* 和を表す変数 s の初期化 *)
    for i:=1 to N do        (* 和を求める主ループ *)
    begin
        s:=s+1.0/i/i;       (* 第 i 項を加える *)
    end;
    writeln('Result : ',s:18:15);   (* 結果の出力 *)
end.
```

※ Pascal 言語では, (* と *) の間に挟まれた部分は注釈となる.

上のプログラムをエディターで書いたら, 保存し,
`gpc num2-1.p -o num2-1` でコンパイルすると,
実行可能 (executable) なファイル `num2-1` ができる.
`num2-1` で実行する.

# 実際の言語による実現例 3 — FORTRAN 言語



FORTRAN のコード (F77) num2-1.f

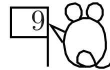
```
␣␣␣␣␣␣PROGRAM SERIES
␣␣␣␣␣␣DOUBLE PRECISION S
␣␣␣␣␣␣WRITE(*,*)'Give N : '
␣␣␣␣␣␣READ(*,*)N
␣␣␣␣␣␣S=0.0D0
␣␣␣␣␣␣DO 100 I=1,N
␣␣␣␣␣␣    S=S+1.0D0/I/I
␣␣100␣CONTINUE
␣␣␣␣␣␣WRITE(*,200) S
␣␣200␣FORMAT(1H ,F22.15)
␣␣␣␣␣␣END
```

- FORTRAN はコンピュータで記号が沢山扱えない 1950 年代にできた古い言語なので，英大文字と数字，少数の記号だけで書かれてきた．
  変数名は**英数字** (英大文字で始める) の 6 文字以内に限られた．
- プログラムを1 行ずつカードにパンチしてコンピュータに入力したので，
  一行 80 カラム，指令は 72 カラム目までに納める，
  最初の 5 桁は行番号, 次の 1 桁は継続行の指示のため，普通は空けておく．
- **暗黙の型宣言**: I ∼ N で始まる変数を 整数変数として，
  A ∼ H, O ∼ Z で 始まる変数を 単精度実数変数として，
  使う 場合は，変数宣言をしなくてよい．
  ⟹ 綴りを間違えても，別の変数と解釈され，エラーにならない!

**コンパイルの仕方**: 上のソースファイルを num2-1.f という名前で作ったら，

```
g77 num2-1.f -o num2-1
```

で実行ファイル `num2-1` ができる．



本日の実習では, 上のどれかのプログラムを用いて,
N をいろいろ替えて出力値を見る.

**いきなり大きな $N$ を入力してはいけない!**
**理由:**
① 時間がかかり過ぎて, 計算が終わらないかもしれない.
② 答が得られても信用できるかどうか不明.

戻ってこなくなったときの止め方:
① Ctrl-C を入力する.
② これがトラップされているときは, 別の kterm を開き,
ps 指令で当該プロセスの番号 XXXXX を得て kill XXXXX を実行.

普通は，結果が変わらなくなったら収束したとみなす.
この判定法は**非常に危険だが，常用されている.**

cf. $\displaystyle\sum_{i=1}^{\infty}\frac{1}{i}$ を収束と判定してしまうかもしれない.
数学の知識または動物的 (工学的) 直観で，誤った判断を防ぐ必要がある.

## 打ち切り誤差 (公式誤差)



**丸め誤差**: コンピュータが実数を有限小数に変換する過程で生ずる
**打ち切り誤差**: 無限に続く式を，計算のために途中で切ったとき生ずる

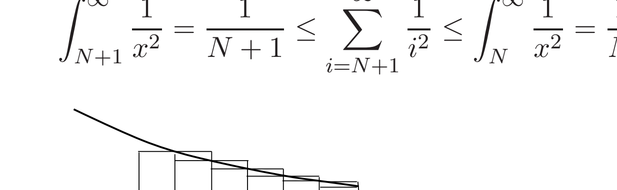
打ち切り誤差は理論的に見積もることが可能な場合もある．

例: $\displaystyle\sum_{i=1}^{\infty}\frac{1}{i^2}$ を $\displaystyle\sum_{i=1}^{N}\frac{1}{i^2}$ で近似したとき，打ち切り誤差は

$$\sum_{i=N+1}^{\infty}\frac{1}{i^2} \le \sum_{i=N+1}^{\infty}\frac{1}{i(i-1)} \le \sum_{i=N+1}^{\infty}\left(\frac{1}{i-1}-\frac{1}{i}\right) = \frac{1}{N}$$

逆の評価も
$$\sum_{i=N+1}^{\infty}\frac{1}{i^2} \ge \sum_{i=N+1}^{\infty}\frac{1}{i(i+1)} \ge \sum_{i=N+1}^{\infty}\left(\frac{1}{i}-\frac{1}{i+1}\right) = \frac{1}{N+1}$$

別解:
$$\int_{N+1}^{\infty}\frac{1}{x^2} = \frac{1}{N+1} \le \sum_{i=N+1}^{\infty}\frac{1}{i^2} \le \int_{N}^{\infty}\frac{1}{x^2} = \frac{1}{N}$$




- この級数の真の値は $\dfrac{\pi^2}{6}$ であることが知られている．

  これを仮定して，上の誤差評価がどのくらい正確か調べてみよ．

参考: $\pi = 3.14159265358979323846264338327950288419716939937510$
C 言語では $\pi$ の値は小数点以下 20 桁が `M_PI` という名で
math.h にマクロ登録されている．

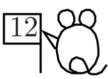
**Taylor 展開による指数関数の計算** $e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}$ の近似計算. 11

C のコード num2-2.c

```
#include<stdio.h>
#include<stdlib.h>
int main(void) {
    int i,N;
    double x,s,t;
    printf("Give x : ");
    scanf("%lf",&x);
    printf("Give N : ");
    scanf("%d",&N);
    s=(double)1;          /* 和の初期値を 0 でなく 1 に */
    t=(double)1;          /* 一般項の初期設定 */
    for (i=1;i<=N;i++) {
        t=t*x/i;          /* 一般項の次数を進める */
        s=s+t;            /* それを加える */
    }
    printf("%22.15lf\n",s);
    return 0;
}
```

- 階乗函数 `fac(i)` を別に作り `s=s+x^i/fac(i)` と書きたいだろうが,
C には冪乗の演算子は無い (ライブラリ函数 pow(x,i) なら有る)
fac(i) を整数函数として再帰的定義で作ると, 非常に小さい i で
オーバーフローが起こり, 使えなくなる.
級数の和の計算においては, 一般項を一々計算するのではなく,
上例のように, 順次更新するようにプログラムするのが常識である.

## 打ち切り誤差の評価




Taylor 展開の剰余項を用いて行う．

$$e^x = s_N + R_N,$$

$$s_N = \sum_{i=1}^{N} t_i, \quad t_i = \frac{x^i}{i!}, \quad R_N = \frac{x^{N+1}}{(N+1)!} e^{\theta x} \quad (0 \le \theta \le 1).$$

$$\therefore \quad |R_N| \le |t_{N+1}| e^x \le |t_{N+1}|(s_N + |R_N|)$$

$$|R_N| \le \frac{|t_{N+1}| s_N}{1 - |t_{N+1}|}.$$

$N$ が大きいとき，$|t_{N+1}|$ は非常に小さいので，上はほぼ

$$|R_N| \le |t_{N+1}| s_N$$

と思って良い．つまり，最後の計算結果に次の一般項を掛けたものがおおよその打ち切り誤差の評価となる．

※ いろいろな $x, N$ について，ライブラリ函数に含まれる
`exp(x)` の値と比較してみよ．
(比較のやり方については，次のプログラム見本を参照せよ．)

**Taylor 展開による三角関数の計算** $\sin x = \sum_{i=0}^{\infty} \frac{(-1)^i) x^{2i+1}}{(2i+1)!}$ の近似計算. 13

C のコード num2-3.c

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
int main(void) {
  int i,N;
  double x,s,t;
  printf("Give x : ");
  scanf("%lf",&x);
  printf("Give N : ");
  scanf("%d",&N);
  s=(double)0;
  t=(double)x;                  /* 一般項の初期設定 */
  for (i=0;i<=N;i++) {
      s=s+t;                    /* 一般項を加えた後, */
      t=-t*x*x/(i+i+2)/(i+i+3); /* その次数を進める */
  }
  printf("Calculated value: %22.15lf\n",s);
  printf("Value of library function: %22.15lf\n",sin(x));
  return 0;
}
```

※ いろいろな $x, N$ についてこれを実行し, ライブラリ函数に含まれる `sin(x)` の値と比較してみよ.

※ 交代級数のプログラミングは上と同様にできる.
FORTRAN の場合は，冪乗演算子 `x**n` が有るが，
一般項を `T= (-1)**(2*I+1)*x**(2*I+1)/FAC(I)` などと書くと
たとい計算できても非常に遅くなるので絶対避けるようにと教えられた.

● C 言語で `exp`, `sin` などの数学函数を使うときは，ソースに
`#include<math.h>` を記すだけでなく，コンパイル時に

```
gcc num2-3.c -lm -o num2-3
```

と，これらの函数を含むライブラリファイルをリンクしなければならない.
`-lm` は,“ 然るべきところ (ライブラリの標準サーチパス) にある ”
`libm.a` あるいは `libm.so` などの略で,
一般的には `/usr/lib/libm.a` などがこれに相当する.
(オタク向け) 君達の使っている環境で `math.h`, 及び
`libm.a` あるいはその代替物で `exp`, `sin` などの函数を含むものが
どこに有るか探してみよ.

**交代級数の収束判定法と打ち切り誤差評価**

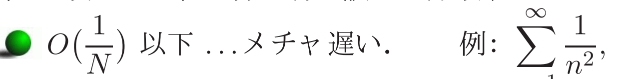
交代級数 $\displaystyle\sum_{i=1}^{\infty}(-1)^{i-1}a_i$ において，$a_i \searrow 0$ なら，級数は収束し，

しかも部分和 $\displaystyle\sum_{i=1}^{N}(-1)^{i-1}a_i$ の打ち切り誤差は $a_{N+1}$ 以下である.

## 級数の収束の速さと加速法



第 N 項までの和の打ち切り誤差で分類する．



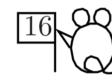
- $O(\frac{1}{N})$ 以下 ...メチャ遅い． 例: $\sum_{n=1}^{\infty}\frac{1}{n^2}$, $\sum_{n=1}^{\infty}\frac{(-1)^{n-1}}{n}$ など
- $O(\frac{1}{N^k})$ $(k>1)$ ...遅いが，$k$ が大きければまあ我慢できる．

  例: $\sum_{n=1}^{\infty}\frac{1}{n^3}$ など（この値は無理数であることしか分かっていない．）
- $O(\frac{1}{a^N})$ $(a>1)$...実用的に速い．

  例: 等比級数 $\sum_{n=0}^{\infty}\frac{1}{2^n}$, それに近い $\sum_{n=1}^{\infty}\frac{n}{2^n}$ など
- $O(\frac{1}{a^{N\log N}})$ $(a>1)$...非常に速い． 例: $\sum_{n=0}^{\infty}\frac{x^n}{n!}$, $\sum_{n=0}^{\infty}\frac{x^{2n+1}}{(2n+1)!}$ など．
- $O(\frac{1}{a^{N^k}})$ $(a>1,\ k>1)$...猛烈に速い．
- $O(\frac{1}{a^{b^N}})$ $(a>1,\ b>1)$...超猛烈に速い．(Newton 法は $a=e, b=2$ に相当)

加速法: 収束の遅い級数は，変形して収束を速くしてから計算する．
18 世紀に盛んに研究された．コンピュータが速くなると，単独計算では
あまり有難味がない．しかし，級数の和でライブラリ関数を作るときは，
少しでも速い方がよいので，今でも重要．
級数の加速法と初等函数の高速計算法は，後の方で再考するであろう．

**加速法の例** $\zeta(2)=\sum_{n=1}^{\infty}\frac{1}{n^2}$ を加速する．




既に見たように，$\sum_{n=1}^{\infty}\frac{1}{n(n+1)}=1$ は既知． また高校で学んだように，

$$\sum_{n=2}^{\infty}\frac{1}{(n-1)n(n+1)}=\sum_{n=2}^{\infty}\frac{1}{2}\Big(\frac{1}{n-1}+\frac{1}{n+1}-\frac{2}{n}\Big)=\frac{1}{2}\Big(1-\frac{1}{2}\Big)=\frac{1}{4}$$

故に，

$$\zeta(2)-1=\sum_{n=1}^{\infty}\Big(\frac{1}{n^2}-\frac{1}{n(n+1)}\Big)=\sum_{n=1}^{\infty}\frac{1}{n^2(n+1)},$$

$$\zeta(2)-1-\frac{1}{4}=\frac{1}{2}+\sum_{n=2}^{\infty}\Big(\frac{1}{n^2(n+1)}-\frac{1}{(n-1)n(n+1)}\Big)$$

$$=\frac{1}{2}-\sum_{n=2}^{\infty}\frac{1}{n^2(n^2-1)},$$

$$\therefore\quad \zeta(2)=\frac{7}{4}-\sum_{n=2}^{\infty}\frac{1}{n^2(n^2-1)}$$

これで $O(\frac{1}{N^3})$ の収束が保証される．

本日の講義内容の自習課題



(1) 数値計算のできるインタープリタを使ってみる – その1

**Python** (UNIX 配布パッケージや Cygwin に標準で含まれる)

```
g0620549$ python        (起動コマンド. この後沢山のメッセージが出るが省略)
>>> 2+3*4+5**2          (Python のプロンプト >>> に計算式を打ち込む)
39                      (** は冪乗, 演算の優先順位は R と同じ.)
>>> sin(1)
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
NameError: name 'sin' is not defined
>>> from math import *    (この準備をしないと関数電卓にならない)
>>> sin(1)
0.8414709848078965        (三角関数の引数はラジアン)
>>> sin(pi/4)             (円周率は pi で呼び出せる)
0.70710678118654746
>>> sqrt(2)/2
0.70710678118654757       (両者の値が一致しない理由を考えてみよ)
>>> s=0
>>> for i in range(1,10000) : s=s+1.0/i/i
...                       (この記号が出たら単に改行する)
>>> s
1.6448340618480652
```

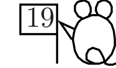
終了指令は Ctrl-D (コントロールキーを押しながら d を押す)
上向き矢印キー↑で以前使ったコマンドが呼び出せ (ヒストリー機能),
左右の移動キー←→で途中を書き換えて訂正や再利用できる (行編集機能)

(2) – その2 Risa/Asir フリーソフトだが, インストールが必要. 18

5 階は export PATH=${PATH}:/home/isstaff/kanenko/Risa/bin
で, 僕がインストールしたものが使えるようになる.

```
g0620549$ asir        (起動コマンド . この後の起動メッセージは省略)
[0] 1/3;              (asir のプロンプト [n] に計算式を打ち込む. 行末に ;)
[1] 1.0/3;
0.333333
[3] @@-0.333333333333333;   (@@ は直前の出力結果を表す.)
3.33067e-15           (表示では単精度に見えるが, 内部では倍精度)
[4] 1/2+1/3;
5/6
[5] 2^32;
4294967296
[6] 2^100;            (出力結果は略)
[7] fac(13);          (これは何でしょう？)
6227020800
[8] fac(1000);        (出力結果は略)
[9] sin(@pi/4);              (π は @pi で表現)
sin(1/4*@pi)                 (デフォールトではシンボルとして扱う)
[10] eval(@@);               (直前の出力結果 @@ を数値に直す)
0.7071067811865475243        (通常の倍精度)
[11] ctrl("bigfloat",1);     (多倍長計算を標準とする)
1
[12] setprec(100);           (多倍長を 100 桁に指定)
105                          (実際の出力桁数)
[13] 1.0/3;           (出力結果は略)
[14] sin(@pi/4);
sin(1/4*@pi)
[15] eval(@@);        (出力結果は略)
[16] quit;            (終了指令)
```

@n で第 n 出力値を引用できる.
C 言語に似た文法でプログラミングができる.
現在インストールされているものにはヒストリーや行編集機能は無いので
X の標準コピペ機能で代用してください.



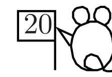

(3) num2-1.c のソースを 読み, コンパイルして実行してみる.

(4) num2-1.f のソースを 読み, コンパイルして実行してみる.

(5) num2-2.c のソースを 読み, コンパイルして実行してみる.
指数函数の近似値を 与えていることを 確認し 観察し ,
誤差を 観察せよ.

(6) num2-3.c のソースを 読み, コンパイルして実行してみる.
$\sin x$ の近似値を 与えていることを 確認し 観察し , 誤差を 観察せよ.
特に, $x = 20$ とかを 代入したときの出力値を 観察してみよ.
観察された 現象の説明を 試みよ.

## 本日の範囲の試験予想問題




**問題 2.1** 無限級数 $\sum_{n=1}^{\infty} \frac{1}{n^2}$ の和を求める次の C プログラムの誤りを指摘し，正しく直せ.

```
int main(void){
  int i,n=1000;
  double s=(double)0;
  for (i=1;i<=n;i++){
    s=s+1/i^2;
  }
  printf("%ld\n",s);
  return 0;
}
```

**問題 2.2** 無限級数の近似和を計算するときに考慮しなければならない誤差にはどのようなものがあるか？

**問題 2.3** 問題 2.1 のプログラムを正しく直して実行したとき，

(1) 級数の近似値としてどのくらいの精度の値が求まるか？

(2) $n$ の値を大きくしたとき，どのあたりまでは近似値が真の値に近づくと期待されるか？

(3) その限界を越えてより良い近似値を求めるための工夫を一つ記せ.

**問題 2.4** 級数 $\displaystyle\sum_{i=1}^{N}\frac{1}{i^2}$ において，$N=10^n$ とし，$n$ を種々の値に取って実験すると，$\displaystyle\sum_{i=1}^{\infty}\frac{1}{i^2}$ の真値 $-\dfrac{1}{N}$ と $O\left(\dfrac{1}{N^2}\right)$ でしか違わないこと，i.e. 近似値は小数点以下第 $n$ 桁目が狂っているが，そこに 1 を足せば，ほぼ $2n$ 桁の近似値が得られることが num2-1.c などで観察される．この理由を数学で説明せよ．

21

**問題 2.5** (1) $\log(1+x)$ の Taylor 展開を計算するプログラムを C 言語で書け．ただし，引数 x をとり，計算値を返す関数だけでよい．

(2) この級数は $x=1$ でも収束し $1-\dfrac{1}{2}+\dfrac{1}{3}-+\cdots+(-1)^{N-1}\dfrac{1}{N}+\cdots=\log 2$ が成り立つことを，$\dfrac{1}{1+x}$ の積分を用いて示せ．

(3) 定積分の理論を思い出して（あるいは高校生的技法により）左辺の打ち切り誤差は $\Theta\left(\dfrac{1}{N}\right)$ であることを示せ．ここに，$f(n)=\Theta\ (g(n))$ は，適当な正定数 $c_1,c_2>0$ により $c_1g(n)\le f(x)\le c_2g(n)$ が成り立つことをいう．

(4) 最終項の分母の $N$ を $2N$ に変えると，誤差が $O\left(\dfrac{1}{N^2}\right)$ に改良されることが実験で確かめられる．この理由を数学で説明せよ．