19980630–0705;0908–20;23–
30;1021;27;1111;



第 V 章担当：服部哲弥

# 第 V 章

# １次元ランダムウォーク（数値計算入門）

## V.1　授業時間とレポート

### V.1.1　授業時間配分表

| | 10 月 20 日 | 10 月 27 日 | 11 月 03 日 | 11 月 10 日 |
|---|---|---|---|---|
| 学籍番号下１桁 | 実習１<br>＋宿題１ | 実習２<br>＋宿題２ | 宿題２<br>＋ 実習３ | 実習３ |
| 奇数 | A | B | 休日 | C |
| 偶数 | B | A | 休日 | C |

| | |
|---|---|
| グループ | 計算機室優先利用（残りの時間は各自で計算機以外の問の解答，プログラム作成，<br>レポートの下書き，などを行う） |
| A | 13:10–14:45 |
| B | 14:45–16:20 |
| C | 最終回につき混雑が予想されるので計算機室の空いている日時に<br>前もってレポートを提出しておき当日は出席不要とするのが望ましい. |

### V.1.2　実習内容

**実習１：** 復習とレポート提出の練習.

- C プログラミングと LaTeX レポート作成の復習.
- レポート提出方法 (web への転送と BBS への投稿) の練習かつ実践.

**宿題１：** 実習２の数学部分の演習解答，実習２のプログラム作成，実習２の LaTeX レポート原稿作成開始.

**実習２：** 実習２のプログラム実行，結果解析，レポート完成，レポート投稿.

- $[0, 1)$ に値をとる一様乱数の統計学の基礎の基礎.

**宿題２と実習３：** 宿題１と実習２にそれぞれ対応.

- １次元ランダムウォーク（$\pm 1$ に値をとる乱数の和）の数学の計算機実験のこと始め.

**成績はこのプリントの最後にある３回のレポートの提出による．提出方法は V.8.4 節 による．**

### V.1.3　各実習の流れ

**宿題** (1) 数学の勉強.
(2) C 言語プログラムの完成.
(3) レポートの下書き開始.

**実習** (1) プログラムのコンパイル，実行，修正 (V.8.1 節).
(2) 結果分析，レポート内容の完成.
(3) レポート (LaTeX 文書ファイル) の作成 (V.8.2 節, V.8.3 節).
(4) レポートの LaTeX コンパイルと修正 (V.8.3 節).
(5) レポート (LaTeX ファイル，dvi ファイル) を web site に転送 (FTP) (V.8.4 節).
(6) 転送したファイルをブラウザ (Netscape) で確認 (V.8.4 節).
(7) レポート提出用 BBS への投稿 (ブラウザ) (V.8.4 節).

**くりかえし** $n$ を $n+1$ に置き換えて，次の宿題と実習に進む (^_^).

## V.2 注意

### V.2.1 一般的注意

- レポート提出方法や使用ソフトは変更がありうる．当日の注意や追加プリントの有無を確かめること．
- 早めにこのプリントを予習し，締め切りや実習日に関わらず，レポートを完成，提出すること．
- 木田先生の担当分のプリントを探しておくこと！
- 佐藤先生の担当分のプリントを探しておくこと！
- １年のときの木田先生の授業（特にブラウザ，ＦＴＰ，テルネット，メール，の意味や使い方）を思い出しておくこと！

(1) この章は１次元ランダムウォークに関する初歩的な数値計算を通して，数値計算という分野を紹介しようというものです．知らない人には恐ろしげな名前かもしれませんが，４年以上で確率論のゼミでもとらない限り数学的なことはできませんので，名前は単なる呪文と思って下さい.

(2) 計算機は初等整数論を含む離散的な量を扱う数学にもっとも向いています．実数を扱う数学はむしろ苦手です．しかし，実用上の需要があまりに大きいため，初期の計算機の用途は実数の近似計算が中心でした．数値計算は計算機実用時代の初期に確立した分野です．この巨大な分野全体を見渡すようなことは**しません**. 乱数をシミュレーションに使うという，極めて基本的だが極めて汎用性の高い数値計算の方法論の一つの入り口を，１次元ランダムウォークという，やっていることは極めて単純だが数学的には極めて高度な問題の入り口に，適用するだけです.

(3) C 言語で説明しますが，他の言語でプログラムを書いてくださっても構いません．例えば荒川先生担当部分で用いた UBASIC と木田先生担当部分で用いた C の両方でプログラムして，結果や計算時間を比べるとたいへん有意義です．また，言語や機械によって結果が変わらないのが計算機の特徴のはずですが，変わる場合のあるところが「数値計算」の**裏**の特徴です.

(4) 以下で「木田プリント参照」とあるのは，木田先生担当の授業の配布プリント「C 言語の基本文法」を指します．C 言語については木田プリントを復習し，横に置いてプログラムして下さい．私は木田プリント（と Turbo C の online help）だけで C プログラムを自作しています.

(5) 早めにこのプリントを予習し，締め切りや実習日に関わらず，解ける課題を解き，レポートすべき内容を下書きし，計算機の空いている時間を利用してレポートを完成して下さい．指定した提出方法は，計算機さえ空いていれば，授業時間外でも提出可能です.
毎度のことですが，実習当日は大勢参加するので計算機が混み合い，試行錯誤する時間がありません．初回

の実習は復習とレポート提出方法の練習が中心なので，内容が物足りないかも知れませんが，各自先の方を予習して下さい．２回目から忙しくなります．

(6) V.8 節 に説明してあるように，レポートは LaTeX による文書を各自の web site に転送し，そのリンクを私の講義ページのＢＢＳに貼ることによって提出して頂きます．以下で「佐藤プリント参照」とあるのは，佐藤先生担当の授業の配布プリント「TeX 入門」を指します．レポート作成のためには佐藤プリントを復習し，横に置いて文書を作成して下さい．LaTeX ソースファイル (report1.tex など) の編集，コンパイルによる dvi ファイルの作成は佐藤プリントを復習して下さい．

(7) V.8.4 節 に説明してあるブラウザ，ＦＴＰ，テルネット，メール，の各ソフトの意味や使い方は１年のときの木田先生の授業で習ったはずです．忘れた人は復習しておいて下さい．これができないと単位がつきません．

### V.2.2　参考書

このプリントは木田プリントと佐藤プリント以外は何も参考にしていません．

数値計算に関しては本はたくさん出ていますが，私は我流なので参考書の良い提案はありません．高校時代に習ったニュートン法と数値積分の台形公式がいちばん役に立っています．実際，

- 広松恒彦・山口和子「アルゴリズムとデータ構造」日本経済新聞社，情報処理技術者試験 [新２種重点ガイド３]，1996 年．

の 2.5 節の表題が「数値計算」ですが，ニュートン法と台形公式が中心です．各自具体的に必要になったときに必要事項の書いてある本を選べばよいと思います．

一様乱数に関しては，木田プリント XI–31 脚注に参考文献が載っています．先ずはそれを勉強して下さい．私は

- B. D. Ripley, *Computer generation of random variables: A tutorial*, International Statistical Review **51** (1983) 301–310.

という論文を人に教わりましたが，これが特に優れた解説とも思えません．むしろ，そこに引用されている

- D. E. Knuth, *The art of computer programming,* **2***, Seminumerical algorithms*, 2nd. ed., Addison–Wesley.

が基本だと推測します．引用は第２版となっていますが，たしか第３版が出ています．翻訳も出ています．このシリーズは乱数に限らず計算機科学の「バイブル」です．著者は TeX の著者でもあって，佐藤プリントに紹介されています．

一様分布以外の分布に従う乱数もあって，むしろ，実用上は一様でない分布のほうが多いです．これらは理想的な一様乱数（を生成するプログラム）が与えられたとき，それを用いてほしい分布に従う乱数を求めよ，という問題として扱われるので，一様分布とは別の研究分野になっています．

- L. Devroye, *Non-uniform random variate generation*, Springer.

が「バイブル」ですが，私が注文したときは絶版でした．原典が絶版になるくらいだから日本語訳は出ていないでしょう．縁があって私も研究したことがあります：

- T. Hattori, H. Nakajima, *Improvement of efficiency in generating random* $U(1)$ *variables with Boltzmann distribution*, Journal of Computational Physics **121** (1995) 238–245.

が，この論文が特に基本論文だというわけではありません．

ランダムウォークやその連続極限としてのブラウン運動は数学的にも物理学的にも非常に重要な研究対象です．数学では確率過程論という分野の出発点であり，かつ，今でももっとも重要な典型例です．確率過程論の教科書は大抵ブラウン運動から始まりますが，先に確率論の初歩を勉強しないと読めないと思います．確率論の初歩については私の home page

`http://150.93.96.124/math/hattori/hattori.htm`

から　／講義のページ／確率論２（信州大学集中講義）　を開いてもらえれば参考文献等の入った dvi file がダウンロードできます.

- 西尾真喜子「確率論」実況出版，1978.

が確率論の初歩の日本語の教科書としてはお薦めです．この教科書の第６章 §3 後半にさりげなくランダムウォークが定義されています.（ランダムウォークに重点をおく教科書の書き方はたぶん日本では新しい可能性だと思う．英語では良書がもっとある.）比較的新しい教科書として

- 小谷真一「測度と確率２」岩波講座現代数学の基礎，岩波書店，1997.

を挙げておきます．その巻末の参考書を見れば比較的新しい教科書がいくつか分かります.

#### V.2.3　謝辞

プログラム作成にあたり津田稔朗君（1998 年大学院修士１年）の多大の協力を得ました．また，レポート投稿用 web フォームと BBS の作成にあたり，斉藤友行君（1996 年入学）と内田由美子さんの多大の協力を得ました．記して感謝します.

## V.3　乱数という概念

### V.3.1　復習

木田プリント XI–31 ページ（最後のほう）:

> ゲームのプログラムを作ったりするには乱数を作ること（サイコロを振るのが代表的）が欠かせません.

サイコロのような「人にとって規則性が見えない」現象を計算機で研究する（あるいは，ゲームソフトの場合，利用する）手段が計算機の発生する乱数（擬似乱数）である.

**授業の実習やレポートを急ぐ諸君は以下をとばして V.4 節 に進んで構わいません.**

### V.3.2　定義（は，ない！）

乱数の「気持ち」は「でたらめな順序で並んだ数の列」である．しかし，この「気持ち」をそのまま丸ごと認めた乱数の定義はありえない．無限列しか考えないならば定義できるかもしれないが，有限列では根本的に不可能なことが次の古典的な笑い話で分かる.（しかも，人や計算機が扱うのは有限列である.）

> ある実験者が 0 と 1 からなる長い乱数列を用意しろと統計学者に言った．統計学者が用意して表を持っていったところ,「５個の数字が必要だったが，列の最初の５個は 11111 だった．これでは乱数にならない」と言われて怒られた．そこでもう一度乱数列を用意して持っていったところ，今度は「１億個の数字が必要だったが，この中に一度も 1 が５個連続して出てこない」と言われて怒られた.

### V.3.3　私見　－　疑似乱数で何が悪い？

木田プリントに解説してあるように，通常用いられている範囲では，計算機の発生する乱数は，でたらめどころかたいへん簡単な規則で作られている．そこに注意して，専門家は計算機が発生する乱数を擬似乱数と呼ぶ.しかし,（乱数の専門家の意見は違うかも知れないが，私の印象では）「計算機が発生する数列は全て規則的（定義の明確な，人が指定したアルゴリズムに基づいている）であるのに対して，サイコロなどは質が違う」という

議論で疑似乱数か否かの区別を強調するのは，意味がないと思う．前小節の笑い話のように有限列に対して全くでたらめな数列というのはありえない．むしろ，用途を指定して初めて「必要なでたらめさ」が議論できる，と思う．各用途毎に，乱数のどういう性質を利用しようとしているのか決まっているはずであり，その性質を満たすものであれば，用途から見たときにでたらめであるという意味で，乱数と呼べる．

### V.3.4　乱数はゲームのためだけか？

木田プリントではじゃんけんゲームが乱数の例として取り上げられている．勉強や，将来は仕事や研究以外で，目にする乱数はゲーム（賭事を含む）だけかもしれないが，仕事や研究のようなまじめなところでも乱数は重要な用途を持っている．

**予測不能な現象のシミュレーション（数値実験）：** 正解の予想がつかないときや最良の解法が分からないときにまず見当を付ける．複雑な現象をシミュレーションによって計算機上に再現することで，到底数学的に解析できないように思える複雑な現象を詳しく調べて隠れた規則性を見いだそうとする．

**数値積分（モンテカルロ法）：** 乱数が持つ「でたらめさ」は，確率変数のサンプルととらえることで well-defined な量を定義しうる．これを逆に用いることで，数学や他の自然科学の厳密な量を乱数を用いて統計学的な近似値を求めることができる．その典型が関数の積分と確率変数の期待値が同じ意味であることを用いたモンテカルロ数値積分である．

### V.3.5　計算機はでたらめに計算できない

「でたらめさ」に関して以上とは別の論点もある．計算機は基本的に与えられた規則に忠実に計算する．だから，乱数といえども初期条件（V.4.1 節 に出てくる例では `seed` という変数に与えられた整数値）を決めると乱数が決まり，何度でも同じ乱数列を発生させることができる．一見，でたらめな数列という乱数の「気持ち」に反するようにみえるが，同じ数列を繰り返せるということはプログラムや得られた結果に誤りがないか繰り返し検査できるという重要な利点がある．（さいころのように同じ数列を作り出す方法がないと，昔作った数列が本当にそのさいころを使ったかどうか確かめようがなくなる．なお，`seed` をわからないようにする方法はある．木田プリント XI–32 ページのプログラム 11.2 の `main` 関数の 2 行目を参照.）

## V.4　一様乱数列の発生

**ここからプリントの最後のレポート課題第１回の作業が始まります．プリントの最後のレポート課題をにらみながら読み進めてください．**

$[0,1)$ 上の一様（疑似）乱数を生成するアルゴリズム（あるいはプログラム）とは，全ての $i = 1, 2, 3, \cdots$ に対して $0 \leq x_i < 1$ を満たす実数列 $x_1, x_2, \cdots$ を（任意の指定した個数）生成するアルゴリズムであって，$[0,1)$ 上の一様分布（ルベーグ測度）に従う独立同分布確率変数列 $X_1, X_2, \cdots$ に対して確率 1 で成り立つ性質は，計算機の実数の精度の範囲で，かつ，乱数の使用目的にとって必要な範囲で，列 $\{x_i\}$ に対しても成り立つものを言うことにする．

この「定義」は通常使われる定義と若干違うが，それは本質的な差ではない，と思う（間違っていたらごめん）.「使用目的にとって必要な範囲で」となっているためにまだ定義になっていないが，V.5 節 で扱う量は「どんな使用目的にとっても必要な範囲」に含まれる．独立同分布確率変数列という言葉は確率論を習うまでは分からないかも知れないが，以下で行う「理論的計算」が成り立つような列とだけ思っていれば今は十分である．

早速 $[0,1)$ 上の一様（疑似）乱数を生成するプログラムを作ろう．

### V.4.1 ライブラリ関数 rand と srand

木田プリント XI–31 ページに従って，ライブラリ関数 rand と srand を用いて $[0,1)$ に値をとる一様乱数を発生させる．関数 rnd が呼び出す毎に乱数を 1 つ与える．関数 rndinit は初期値 seed と乱数を呼び出す回数 n をキーボードの入力から取り込む．主関数 main は乱数を単に n 回呼び出して画面表示する．

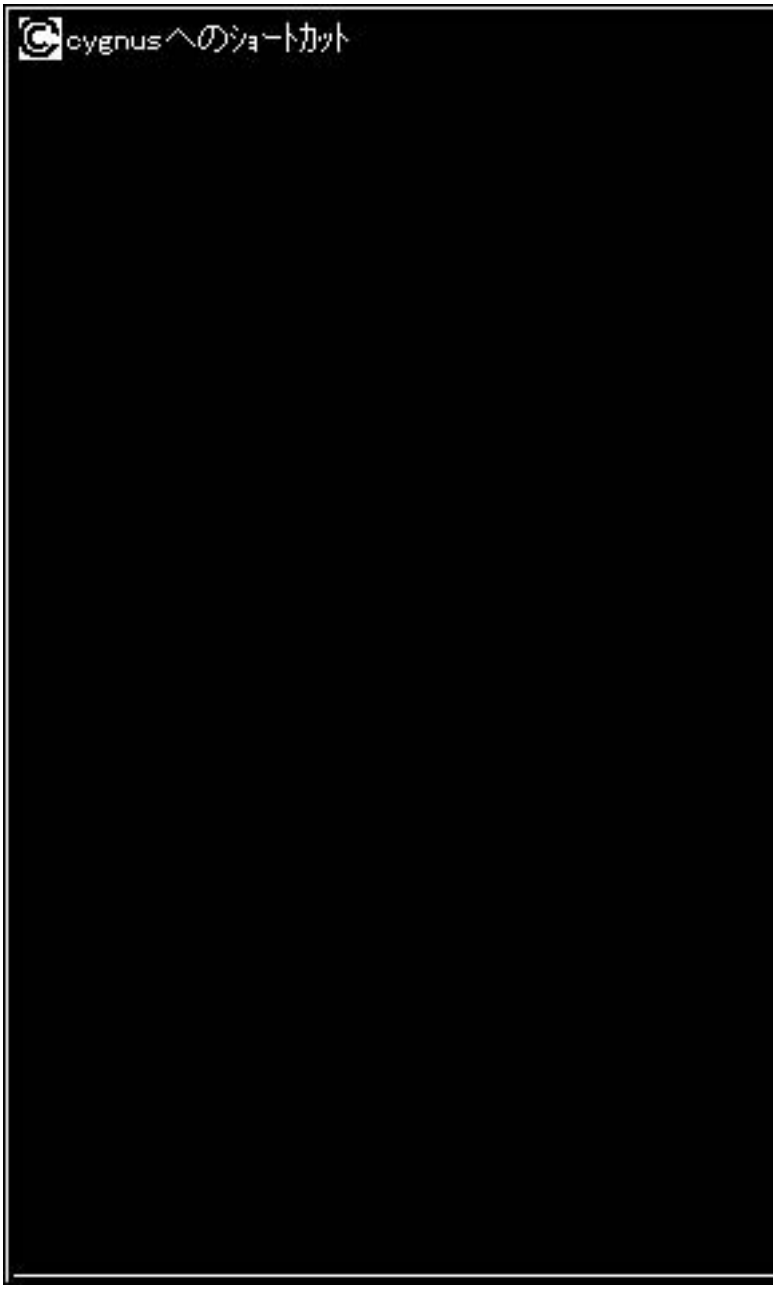
```
/* random.c ( [0,1) random number) */

#include

#include <math.h>
#include <stdlib.h>

double rnd      (void);
void   rndinit  (int *iteration);

void main(void)
{
 int i, n;
 printf ("\n[0,1) uniform random number. \n");
 rndinit (&n);
 for (i=1;i<=n;i++) printf ("%15.12f\n", rnd ());
}

double rnd (void)
{ return (double) rand () /( (double) RAND_MAX + 1.0 );}

void rndinit (int *iteration)
{
 int n, seed;
 printf ("Input seed: ");  scanf ("%d", &seed);
 printf ("Input sequence length: ");  scanf ("%d", &n);
 if (seed <= 0) seed= 1;
 srand (seed);
 *iteration=n;
}
```

以上のプログラムを WZ で作成し（例えば） random.c という名前で保存する．V.8.1 節 を参照して，gcc random.c -o random.exe でコンパイルし，./random.exe で実行する．

実行すると seed （タネ）と sequence length （列の長さ）を入力するよう求められる（画面に表示が出る）．

seed は乱数の初期値を決める整数.（レポートで報告するときの初期値は課題 1 に指定してある.）「列の長さ」n は乱数列を何個打ち出すかを与える．画面は２５行以内なので，最初のうちは 20 くらいまでにしてみるといいかもしれない．

正しくプログラムが実行できれば，次のような出力を得るはずである．




関数 rand は 0 以上 RAND_MAX 以下の整数に値をとる乱数であり，関数 srand はその初期値を与える．$0 \leq x < 1$ に値をとる関数にするために，rnd は rand の値を RAND_MAX+1 で割っている．RAND_MAX の値は次のようなプログラムを実行すれば分かる．

```
#include<stdio.h>
#include<stdlib.h>
void main(void){printf("RAND_MAX=%10d",RAND_MAX);}
```

用いるコンパイラによって値が違うが恐らく実際の数値は $2^{15}-1$ か $2^{31}-1$ になる．よって rnd の精度もせいぜいこの逆数程度であることが分かる．

**レポート課題第１回の作業はここまで．引き続き宿題１の範囲が始まります．宿題１は第２回授業以前にすませて下さい．レポート課題第２回の一部として提出することになります．計算機室が空いていれば，宿題に限定せず第２回課題に取り組んで下さい．**

### V.4.2　合同法による簡単な疑似乱数

rand や srand のようなライブラリ関数を使うとプログラミングは簡単だが，実際に計算機がどういう計算（アルゴリズム）で乱数（疑似乱数）を生成しているかはわからない．実は，合同法と呼ばれる非常に単純なアルゴリズムで疑似乱数を生成できる．この乱数は問題点が指摘されている（乱数らしくない規則性がある）が，多くの実用上の目的のためにはこの一番単純な方法が十分役に立つ．

木田プリント XI–31 ページに紹介されている具体的な例：

```
a=0x41a7; m=0x7fffffff;
とします．初期値 seed を 1 以上 m-1 以下に定め，
   seed= (seed * a) % m; random = (double) seed / (double) m;
をくりかえせば random は 0 以上 1 未満の実数の疑似一様乱数となります．
```

（木田プリントにはキャスト (double) が書かれていないが，seed, m ともに整数で random が実数なので，実数型に明示的に変換して割り算するのがよいでしょう．）

この記述に示されたアルゴリズムをプログラムに翻訳してみよう．(WZ で A:\random.c を開いて，メニューの [ファイル] [名前を付けて保存] で A:\randomb.c などとして名前を付け替えて作業するのが簡単．完成したら保存する．)

**random.c から randomb.c への変更箇所．**

1．表題．

```
/* random.c ([0,1) random number) */

↓

/* randomb.c ([0,1) random number) */
```

2．#include <stdlib.h> と double rnd (void); の間への挿入．

```
#define a   0x41a7
#define m   0x7fffffff
int seed;
```

3．関数 double rnd (void) の定義．

```
{ return (double) rand () /( (double) RAND_MAX + 1.0 );}

↓

{ seed= [          ]  % m; return (double) seed / (double) m; }
```

4．関数 void rndinit (int *iteration) の定義冒頭．

```
 int n, seed;

↓

 int n;
```

5．関数 void rndinit (int *iteration) の定義後半．

```
if (seed <= 0) seed= 1;
srand (seed);
```

↓

```
if (seed <= 0 || seed >= m) seed= 1;
```

空欄は引用した木田プリントのアルゴリズムに一致するように埋める．以上の変更を行うと，一見プリントのアルゴリズム通りになるように見える．しかし，正しく空欄を埋めても，このプログラムが $[0,1)$ 一様乱数を生成しないことは，コンパイルして実行してみればたぶんすぐ気づくと思う．この `randomb.c` はまだ間違っている．

### V.4.3　C の整数型変数

**この V.4.3 節 はもしわかりにくければ，とりあえず最後の文だけ読めば `randomb.c` を完成させることができます．**

何がいけないか？いま考察しているアルゴリズム（に限らず一様乱数のアルゴリズム全て）は登場する整数変数が負の値をとらない．C 言語の整数変数は通常 32 bit（２進数 32 桁）の数を記憶し，上から $i$ bit（２進桁）目は $2^{32-i}$ という整数を表すが，最初の bit ($i=1$) だけは $2^{31}$ では**なく**，**符号**を表す．従って，最初の bit だけ特別扱いしないといけない．合同法のように整数変数同士の積や和をとって再び元の変数に代入する，という操作を繰り返すと，($2^{32}$ 以上になると変数に入らなくなる－ overflow －が，その前に）結果が $2^{31}$ 以上になった時点で結果を正しく代入できなくなる．問題のアルゴリズムは結果が $2^{32}$ 以上にならないようになっているので変数の記憶場所から overflow することはないが，$2^{31}$ 以上になることはある．このとき，

(1) コンパイラがコンパイルの段階で問題ありと指摘する．
(2) コンパイルの時点では気づかないが，`.exe` を実行中に先頭ビットが符号として特別扱いされていることをチェックするルーチンをコンパイラが付け加える．
(3) 先頭の符号 bit を $2^{31}$ という数字のように扱って計算を続ける．

のいずれかが起こる．最初の２つの場合はプログラムを指示通りに完成しても動かない，ということになり，最後の場合は正の数同士のかけ算や足し算で負の数が出てしまう．このように，通常の整数変数（`int` で定義された変数）では正の大きな数に正の数を加えると負になる，ということが起こりうる．

プリントのアルゴリズムの場合は，整数変数の先頭ビットも含めた全ての桁を，正の数を表すことにしておけば解決する．このためには明示的に `unsigned`（符号がない＝先頭ビットを符号ビットと解釈せず $2^{31}$ という数字として扱う）宣言をする必要がある．

言い換えると，V.4.2 節 の `randomb.c` の中のある `int` 宣言文を `unsigned int` 文に換えれば正しく $[0,1)$ 一様乱数を生成するようになって，`programb.c` が完成する．(どの行を換えればよいかは，宿題.)

## V.5　データの統計量の計算

**この節もレポート課題第２回の作業です．課題をにらみながらどんどんプログラムを実行して下さい．レポート課題はプログラム実行だけでなく数学も入っています．忘れないように先にやっておきましょう．**

### V.5.1　度数分布

数列 $x_1, x_2, \cdots,$ が $[0,1)$ に値をとる一様乱数であるために最初に必要な性質は，その**経験分布が（計算機の精度の範囲で）一様分布**であるということである．即ち，$[0,1)$ の任意の $0 \le a \le b < 1$ に対して

$$\lim_{n\to\infty} \frac{1}{n} \sharp\{1 \le i \le n \mid a \le x_i \le b\} = b - a \tag{V.5.1}$$

が成り立つことである．ここで $\sharp A$ は集合 $A$ の要素の個数．(V.5.1) が成り立つとき任意の開集合，閉集合，それらの可算個の共通部分や和集合 $A$ に対して $\lim_{n\to\infty} \frac{1}{n} \sharp\{1 \le i \le n \mid x_i \in A\}$ は $A$ のルベーグ測度（いわゆる「長さ」）に等しいことが知られている．

計算機は有限の量しか扱えないので，極限といっても実際は計算機とコンパイラによって決まる十分大きな $N$ で左辺を考えれば十分だし，計算機の表せる実数は有限の精度しか持たないので等号も計算機とコンパイラによって決まるある誤差の範囲内に（左辺と右辺の差が）あることを表す.（以下この注意を省略する.）

あるデータの経験分布を数値計算で調べる簡単な分析手段として度数分布をとる方法がある．これは $[0,1)$ をある幅の区間 (bin と呼ぶ) に分割して，それぞれの bin に入るデータ数を数え上げて表やグラフにする方法である.（たぶん小学校のとき習ったと思う.）$[0,1)$ を $M$ 等分（幅 $1/M$ の bin に分割）して度数分布を調べるとすると，$x_1, x_2, \cdots,$ が一様分布に従う乱数列ならばどの bin $B$ に対しても，(V.5.1) より，

$$\lim_{n\to\infty} \frac{1}{n} \sharp\{1 \le i \le n \mid x_i \in B\} = 1/M$$

でなければならない．従って $N$ が十分大きければどの bin の度数 $\sharp\{1 \le i \le n \mid x_i \in B\}$ も等しくほぼ $n/M$ になるはずである．これが度数の理論値ということになる．もちろん「ほぼ」等しい，というとき，逆にどれくらいずれていても「変ではない」か，は，(V.5.1) だけからは分からない．$n$ を増やして調べるとき，各 bin の度数とデータ数 $n$ の比が $1/M$ にちっとも近づかないようならどうもおかしいようだ，と言えるだけである．この点については V.6.4 節 でもう一度取り上げる．

度数分布を数値計算するプログラム例 `dosu.c` を作ってみよう．関数 `rnd` と `rndinit` は `random.c` と同じ．配列 `dosu` に各 bin 毎の度数が入る．Bin の個数は変数 `bin`.

```
/* dosu.c (random number; empirical distribution) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define bin 10

double rnd     (void);
void   rndinit (int *iteration);

void main(void)
{
 int dosu[bin]={0,0,0,0,0,0,0,0,0,0}, i, n;
 printf ("\nDistribution of random numbers.\n");
 rndinit (&n);
 printf ("\n n=%8d\n",n);
 for (i=0;i<bin;i++) printf ("  <%3.1f ",((double) i +1.0) /(double) bin);
 putchar ('\n');
 for (i=1;i<=n;i++) dosu[(int) (rnd ()* bin)]++;
 for (i=0;i<bin;i++) printf ("%6d ", dosu[i]);
 putchar ('\n');
}

double rnd (void)             {  random.c に同じ  }
void rndinit (int *iteration) {  random.c に同じ  }
```

seed=178001 のときの実行例を掲げておく．コンパイルしてリダイレクト ./dosu.exe >> dosu.dat （V.8.2 節 参照）で結果をファイルに落としたものを編集した．

```
 n=       1
  <0.1   <0.2   <0.3   <0.4   <0.5   <0.6   <0.7   <0.8   <0.9   <1.0
     0      1      0      0      0      0      0      0      0      0

 n=     100
  <0.1   <0.2   <0.3   <0.4   <0.5   <0.6   <0.7   <0.8   <0.9   <1.0
     8     12      7     12     10      9      6     14     13      9

 n=   10000
  <0.1   <0.2   <0.3   <0.4   <0.5   <0.6   <0.7   <0.8   <0.9   <1.0
  1051    999   1015   1015    993    996    962   1011    993    965

 n= 1000000
  <0.1   <0.2   <0.3   <0.4   <0.5   <0.6   <0.7   <0.8   <0.9   <1.0
 99999  99994 100056 100148  99893  99765  99583 100298 100231 100033
```

### V.5.2　平均と分散

度数分布は bin の数を大きくとれば精密な（細かい情報の入った）統計量であり，乱数の可否を調べるのに適切な量の一つであるが，細かい情報を得ようとして bin の数を増やすとデータの大きさも大きくなり，逆に数値計算のばらつきが大きくて，正しい値に近づいて行くかどうかのチェックがしにくい面がある．

$n$ 個の数の列（大きさ $n$ のデータとも呼ぶ） $x_1, x_2, \cdots, x_n$ が与えられたとき，このデータの簡単な統計量（統計的性質に関する量）として平均値 $E_n$ と（不偏）分散 $V_n$ が良く知られている：

$$E_n = \frac{1}{n}(x_1 + x_2 + \cdots + x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i\,, \tag{V.5.2}$$

$$V_n = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - E_n)^2\,. \tag{V.5.3}$$

積分論より次のことが知られている． (V.5.1) が成り立つとき，

$$\lim_{n\to\infty}\frac{1}{n}\sum_{i=1}^{n} f(x_i) = \int_0^1 f(x)\,dx \tag{V.5.4}$$

が任意の関数 $f$ と任意の乱数列に対して成り立つ．（正しくは， $|f|$ の積分が有限な任意の可測関数に対して確率 1 で成り立つ．可測関数という言葉はまだ知らないかも知れないが，大抵の関数は可測関数である．また，確率 1 で成り立つとは，「でたらめに数列を選んだら必ず成り立つ」という気持ちを厳密に定式化した言葉である．計算機で生成した乱数が「正しい乱数」ならば，誤差の範囲で成り立たなければいけない性質の一つである，ということを意味する．）

(V.5.4) と (V.5.2) と (V.5.3) から $x_1, x_2, \cdots,$ が一様乱数列ならば $E = \lim_{n\to\infty} E_n = \int_0^1 x\,dx$ 及び $V = \lim_{n\to\infty} V_n = \int_0^1 (x-E)^2\,dx$ を得るので，$n$ が十分大きいときの平均値 $E_n$ と不偏分散 $V_n$ はそれぞれ理論値 $E$, $V$ にほぼ等しいはずである．$E, V$ の値はすぐ求まるので宿題にしておこう．

乱数列の平均値を数値計算するプログラム `heikin1.c` を挙げておく．変数 `n` が項数，変数 `sum` は乱数の和 $S_n = x_1 + x_2 + \cdots + x_n$，`mean` が求める平均値 $S_n/n$.

```
/* heikin1.c (average of random numbers) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double rnd     (void);
void   rndinit (int *iteration);

void main (void)
{
 int i, n;  double mean, sum;
 printf ("\nAverage of random numbers.\n");
 rndinit (&n);

 for (sum=0.0, i=1;i<=n;i++) sum+= rnd ();
 mean= sum/(double) n;

 printf ("heikin = %16.14f\n", mean);
}

double rnd (void)             {  random.c に同じ  }
void rndinit (int *iteration) {  random.c に同じ  }
```

$n$ が大きくなると次第に理論値に近づく，ということを確かめるためにはいろんな $n$ で計算してその変化を追わないといけない．これまでのプログラムは $n$ を与える毎に実行し直していたが，手間も時間も無駄である．

$n$ を与えたとき $n$/hyouji 毎の中間結果も含めて平均値と分散を計算するプログラム bunsan.c の例を挙げておく.（例では画面に収まるように hyouji= 15 にとってある.）プログラムは複雑になるが，乱数を用いた計算ではこのような中間集計のプログラム作りは必須である．hyouji が結果の表示行数．Loop 変数（for を回す添え字の役割を果たす変数）gyo はいくつ目の表示に向けて計算中かを示す．mean が平均値，var が分散.

```
/* bunsan.c (random numbers; variance) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#define hyouji 15

void   rndinit (int *iteration);
double rnd     (void);

void main (void)
{
 int gyo, i=0, n;
 double kankaku, mean=0.0, r, var=0.0;
 rndinit (&n);
 printf ("%10s%21s%16s\n", "n", "mean", "var");
 kankaku = (double) n /(double) hyouji;
 for(gyo=1;gyo<=hyouji;gyo++)
  {
   while(i<gyo*kankaku && i<n)
    {
     r= rnd ();
     mean+= (r-mean)/(double) (i+1);
     var+= (r-mean)*(r-mean);
     i++;
    }
   printf("%14d       %16.14f %16.14f\n",i,mean,var/(double) (i-1));
  }
}

double rnd (void)             {  random.c に同じ  }
void rndinit (int *iteration) {  random.c に同じ  }
```

ところで，heikin1.c では，定義 (V.5.2) に従って和 $S_n = x_1 + x_2 + \cdots + x_n$ を計算してから $n$ で割って平均を求めた．これに対して bunsan.c では，平均値 mean を計算するのに mean+= (r-mean)/(double) (i+1) という計算を繰り返している．つまり，次のような漸化式で平均値 $E'_n$ を求めていることになる：

$$E'_{i+1} = E'_i + (x_{i+1} - E'_i)/(i+1),\ i = 0, 1, 2, \cdots, n-1,\ \ E'_0 = 0\,. \tag{V.5.5}$$

これが (V.5.2) の $E_n$ と理論的には同じ値を与えること $E_n = E'_n$ の証明も宿題とする．何故このようなアルゴリズムを考えるかは V.5.3 節 で.

### V.5.3　情報落ちと桁落ち

実数は無限小数と一対一に対応するから，殆どの実数は（何進法で書こうと）計算機上では正確には表現できない.（記憶装置は有限だから仮に小数で表したときある桁から先の情報は切り捨てるしかない.）これは計算機

が人間に比べて劣っているという意味ではない．人間も寿命が有限なの，異なる２つの実数の差を「身を以て」感じることは殆どできない．

現代の計算機は記憶場所を十分大きく確保できるので，普段は実用上問題にはならない（あたかも正確に計算できると思いこんでプログラムしてもそんなに変な結果が出ない）ことが多い．しかし，問題は存在するのであって気をつけないと誤った答えを得ることがある．次のプログラムは実数変数に有効精度があることと，情報落ちと呼ばれる（間違いを引き起こす恐れがある）現象があること，の説明のために無理に作ったものである．

```
/* seido.c (jissuu hensuu no keta-suu) */
#include <stdio.h>
void main(void)
{
 double sum1, sum2, x; int i, n;
 x=1.0e-10;
 for (i=1;i<=10;i++) { printf ("%24.22f\n",1.0+x); x=x/10.0; }
 putchar('\n');

 n=2000000; x=1.0e-16;
 sum1=0.0;  for (i=1;i<=n;i++) { sum1+=x; } sum1+=1.0;
 sum2=1.0;  for (i=1;i<=n;i++) { sum2+=x; } sum2+=0.0;
 printf ("0+%8d*(10^(-16))+1=%24.22f\n",n,sum1);
 printf ("1+%8d*(10^(-16))+0=%24.22f\n",n,sum2);
 putchar('\n');

 n=3000; x=5.0;
 sum1=0.0;    for (i=1;i<=n;i++) { sum1+=x; }
 sum2=1.0e17; for (i=1;i<=n;i++) { sum2+=x; } sum2+=-1.0e17;
 printf ("%10s%8d*%1.0f%10s = %10.0f\n","",n,x,"",sum1);
 printf ("%10s%8d*%1.0f%10s = %10.0f\n","10^(17) + ",n,x," - 10^(17)",sum2);
 putchar('\n');
}
```

コンパイル後リダイレクトして実行 `./seido.exe > seido.dat` すると結果を記録できる．結果は以下のようになるはずである．

最初の一群の数字は 1 に近い数の列であるが，あるところで突然正確に 1 になってしまう．これは実数変数に入っている実数はある桁数（有効精度）より小さいところの情報を切り捨てていることを表す．次の２行の出力は左辺を計算した結果を右辺として表示している．右辺は異なる数値が出力されているが，プログラムを見ると等しい量を計算していることになっている．つまり，計算機が計算の順序によっては間違えることを表す．小さな数の足し算で大きな違いを生む情報落ちと呼ばれる現象である．最後の２行も同様であるが，結果が極端に違っているはずである．情報落ちは気をつけないと結果に重大な影響を及ぼす場合があることの実例である．

少しいかめしく言うと，実数の計算では常に潜在的に次の２種類の精度の減少（誤差の増大）の危険がある．

**情報落ち．** ２つの実数変数にそれぞれ大きな数と小さな数が入っているときの足し算（引き算も）で起こる．両者の比が変数の有効精度よりも本当に小さくなると，小さな数を足した効果は有効精度の範囲外に出るので，結果はゼロを足した（何も足さない）のと同じ効果である．小さな数でもたくさん足せばその合計は有効精度の範囲内に入る．それなのにゼロ扱いになるから誤った答えを与える．つまり情報落ちが起きうるプログラムは間違ったプログラムである．

浮動小数点方式で実数を表現するとき起きうる現象．

**桁落ち．**２つの実数変数にほぼ同じくらいの大きさの同符号の数が入っているときその差をとると起きる．差が，元の数に比べて非常に小さいとき，結果の相対的な精度（有効桁）は下がる．誤りとは必ずしも言えない（そもそもその程度の精度でしか得られない問題を扱っている可能性もある）が，工夫すればもっと精度を上げられるのにそれを怠ったために必要以上に桁落ちが起きうるプログラムならば，それは間違ったプログラムである．

実数の計算機上の表現は有効精度が有限なので，必ず起きうる現象．

heikin1.c のように，同程度同符号の数の和（や平均など）を求める場合に問題になるのは**情報落ち**であり，２つの数を比べるために差をとる，というような計算で問題になるのは**桁落ち**である．

V.5.2 節 の heikin1.c と bunsan.c で扱った２つの平均の計算の結果を直接比べてみよう．

```
/* heikin3.c (comparison of the two methods) */

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

#define hyouji 15

double rnd     (void);
void   rndinit (int *iteration);

void main (void)
{
 int gyo, i=0, n;
 double kankaku, mean1, mean2=0.0, r, sum1=0.0;
 rndinit (&n);
 printf ("%15s%17s%17s\n", "n", "mean1", "mean2");
 kankaku = (double) n / (double) hyouji;
 for (gyo=1;gyo<=hyouji;gyo++)
  {
   while(i<gyo*kankaku && i<n)
    {
     r= rnd ();
     sum1+= r;
     mean2+= (r-mean2)/(double) (i+1);
     i++;
    }
   mean1= sum1/(double) i;
   printf ("%17d    %16.14f %16.14f\n",i,mean1,mean2);
  }
}

double rnd (void)             {  random.c に同じ  }
void rndinit (int *iteration) {  random.c に同じ  }
```

リダイレクト ./heikin3.exe > heikin3.dat でファイルに落とした結果を編集したもの：

```
Input seed: 11011
Input sequence length: 1000000000
             n            mean1            mean2
          66666667   0.49994014721710 0.49994014721697
         133333334   0.49995448104385 0.49995448104369
         200000000   0.49995026176102 0.49995026176090
         266666667   0.49995236090597 0.49995236090588
         333333334   0.49996201440547 0.49996201440526
         400000000   0.49997147761765 0.49997147761743
         466666667   0.49996723199936 0.49996723199893
         533333334   0.49997317312894 0.49997317312863
         600000000   0.49997612805242 0.49997612805224
         666666667   0.49997382145930 0.49997382145926
         733333334   0.49997325794103 0.49997325794115
         800000000   0.49997041307331 0.49997041307362
         866666667   0.49997711473980 0.49997711474012
         933333334   0.49997223323102 0.49997223323127
        1000000000   0.49997533367841 0.49997533367876
```

幸い，seido.c を実行して得たような重大な差はないようにみえる．それを見越して V.5.2 節 の bunsan.c では分散 var のほうは単純なアルゴリズムで平均を計算している（正しい態度ではない）．

情報落ちは２つの実数変数にそれぞれ大きな数と小さな数が入っているときの足し算（引き算も）で起こるが，両者の比が変数の有効精度よりも本当に小さくなったときに初めて影響が顕著になる．現在の通常の C 言語のコンパイラは，seido.c の実行で分かるように，（１０進法で）15 桁程度の有効精度があるので，heikin3.c のように，大きさが 1 くらいの数をどんどん足していって合計が 15 桁を越える（情報落ちが起こり始める）には 1000 兆回程度の足し算を必要とする．こんな巨大な計算はめったにないし，そもそも計算機のスピードがそこまで速くないので，パソコンで情報落ちを確かめるのは現代では難しくなっている．もちろん，情報落ちを気にしなくても大丈夫ということだから，悪いことではない．但し，天気予報や精密科学研究計算，その他実用上の巨大プロジェクト一般でスーパーコンピュータで巨大な計算をする場合には気をつけないといけない．

なお，UBASIC では，ちょっと違う実数変数の取り扱い（浮動小数点方式ではなく固定小数点方式）を行っているので，情報落ちは起こらない．興味のある諸君はぜひ seido.c を UBASIC に翻訳して結果を比べてほしい．

桁落ちは有効精度が有限ならば必ず起きる現象なので，UBASIC でも（その他どんな言語であろうと有限の計算機で実数を扱う限り）桁落ちは起きる．桁落ちの顕著な例は，木田祐司，牧野潔夫「UBASIC によるコンピュータ整数論」，日本評論社，1994 年，第 1 章 5 節（問題）問題 2 にある．

**レポート課題第２回の作業はここまで．引き続き宿題２が始まります．宿題２は第３回授業以前にすませて下さい．次の節 V.6 節 は全て宿題２とレポート課題第３回の作業です．１週間の休みの間に片づけておきましょう．計算機室が空いていれば，宿題に限定せず第３回課題に取り組んで下さい．授業日より以前に計算機室の空いているときにプログラムやレポートをファイルに打ち込んでおきましょう．授業日より前にレポートの完成版を web page に貼って，リンクを BBS に提出すれば，授業に出なくても出席扱いになります．**

## V.6　ランダムウォーク

以下，簡単のために $-1$ または $+1$ が等しい割合で現れる乱数列を考える．硬貨投げ（表，裏をそれぞれ $+1$, $-1$ に対応させる）に相当すると思えばよい．

### V.6.1　乱数の図示

乱数の使い道の重要な一つとしてシミュレーションがあることは V.3.4 節 に書いた．複雑な現象を計算機上で実現して何が起こっているか実際に目で確かめたり，いろんな計算を試してみることで新しい定理を見いだすきっかけの一つにするのである．いわば定理や証明を見つけるための例題の役割である．

乱数をこのような目的に使うときは，結果を図示すること（コンピュータグラフィックス）がとても重要なステップになる．残念ながら授業ではグラフィックスをやる余裕はない．グラフィックスは数値計算と違って計算機の詳細に強く依存するので，機械がちょっと変わるとできないことがあり，授業で導入しづらい面もある．また，研究論文作成などでは計算は C 言語などのコンパイラ言語でやっておいて，出力の長い数値データを `Mathematica` や `gnuplot` などのグラフ処理を意識した高水準言語で処理して作図することのほうが多い．ここでは「星印を打つ」という荒っぽい方法で図示の気分だけ味わってもらう．

先ずは $\pm 1$ を等確率でとる乱数列の図示をしてみよう．乱数列 $x_1, x_2, \cdots, x_n$, はでたらめに並んだ列だから，そのまま図示しても規則性が見つかるはずはない．（見つかったら乱数の資格がない !? ）むしろ，この場合の図示は，規則的に見えたらプログラムが間違っているかも知れない，という逆説的な価値がある．

次の例は $[0, 1)$ 一様乱数に基づいて $\pm 1$ 乱数 `irnd` を `main` 関数の中で与えている．関数 `memori` は星印の位置がどういう値を表すかの目盛りを書く．（このプログラムでは $\pm 1$ の値しかとらないから目盛りはいらないが，以下のプログラムでは必要になる．一般に目盛りをプログラムするのは面倒なためついさぼりがちだが，あとで必ず何を意味する図か分からなくなる．レポートや答案に式だけでなく説明を書くべきであるのと同様，計算結果を図示するときは必ずメモリや図の意味の説明を書くべきである．）`length` は画面横方向の $-1$ と $+1$ の距離を画面の文字数で表した数．これを変えれば表示の幅を変えられる．`iplot` は画面の文字数で表した星印の位置（即ちその時点での $\pm 1$ 乱数の値を表す．この値を関数 `plotstar` に渡すとしかるべき位置に星印を打つ．

```
/* graph.c ({-1,+1} random number) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define length   20
double rnd      (void);
void   rndinit  (int *iteration);
void   memori   (void);
void   plotstar (int n, int i);

void main (void)
{
 int i, iplot, irnd, n;  double r;
 printf ("\nRandom sequence of -1, +1. \n");
 rndinit (&n);
 memori ();
 for (i=1;i<=n;i++)
  {
   r=rnd ();
   if (r < 0.5) irnd= -1;  else irnd= 1;
   printf ("%5.3f     ", r);
   iplot= (int) ( ((double) irnd+1.0)/2.0*(double) length );
   plotstar (length, iplot);  putchar ('\n');
  }
 putchar ('\n');
}

double rnd (void)             {  random.c に同じ  }
void rndinit (int *iteration) {  random.c に同じ  }

void memori (void)
{
 int i;
 for (i=0; i<10; i++) putchar (' ');
 putchar ('-');
 for (i=1; i<length/2; i++) putchar (' ');
 putchar ('0');
 for (i=1; i<length/2; i++) putchar (' ');
 putchar ('1');
 putchar ('\n');
}
```

```
void plotstar (int n, int i)
{
 int j;
 if (i>=0 && i<=n)
 {
  for (j=0; j<i; j++) putchar (' ');
  putchar ('*');
  for (j=i+1; j<=n; j++) putchar (' ');
 }
 else for (j=0; j<=n; j++) putchar (' ');
}
```

画面は 25 行程度しかないので $n = 20$ 程度で実行するか，V.8.2 節 を参考にしてファイル graph.dat に結果を書かせるプログラムに直し，実行が終了してから graph.dat を WZ で開いて眺めるのが良い．実行結果の例を示しておく.（図は画面表示と白黒反転してから bmp に保存したものである.）

```
A:¥zwork¥C>graph

Random sequence of -1, +1.
Input seed: 178001
Input sequence length: 15
          -         0         1
0.187     *
0.510                         *
0.109     *
0.734                         *
0.172     *
0.522                         *
0.044     *
0.751                         *
0.360     *
0.552                         *
0.853                         *
0.964                         *
0.039     *
0.281     *
0.775                         *
```

### V.6.2　ランダムウォークの図示

乱数列 $x_1, x_2, \cdots,$ を図示しても規則性は見えないが，乱数列の最初の $n$ 個の和を $S_n = x_1 + x_2 + \cdots + x_n$ と書くとき，$S_1, S_2, \cdots,$ は， $S_{n+1}$ が $S_n$ と $\pm 1$ しか差がない，という意味で，つながった線を思わせる図ができる．この図形のことをランダムウォークのサンプル（見本）と呼ぶ．ランダムウォークには数学的に独特かつ奥深い重要な性質がある．

次のプログラムの例はランダムウォークを図示する．この例から今まで使ってきた $[0, 1)$ 一様乱数を生成する関数 rnd の代わりに $\pm 1$ 乱数を生成する関数 irnd を使うことにする．graph.c では $[0, 1)$ から $\pm 1$ への翻訳を main 関数でやっていたが，この例以降は irnd 関数の中で $[0, 1)$ 乱数の生成と $\pm 1$ 乱数への翻訳を両方やる

ように変えた．graph.c も同じように修正できるが，宿題としておく．

```
/* rw.c (Random walk - sum of random {-1,+1}) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define length   60
int  irnd      (void);
void rndinit   (int *iteration);
void plotstr_a (int n, int i);

void main (void)
{
 int i, n, sum;
 printf ("\nRandom walk - sum of random sequence of {-1,+1}.\n");
 rndinit (&n);
 for (sum=0, i=1;i<=n;i++)
  {
   sum+= irnd ();
   printf ("%7d   ", sum);
   plotstr_a (length, sum+length/2);
   putchar ('\n');
  }
 putchar ('\n');
}

int irnd (void)
{
 double rnd;
 rnd= (double) rand () /( (double) RAND_MAX + 1.0 );
 if (rnd < 0.5) return -1;  else return 1;
}

void rndinit (int *iteration) {  random.c に同じ  }

void plotstr_a (int n, int i)
{
 int j; char line[length+2]; char *pline;
 pline=line;
 for (j=0; j<=n; j++) *(pline+j)=' ';
 for (j=0; j<=n; j+=5) *(pline+j)='|';
 *(pline+length/2)='0';
 if (i>=0 && i<=n) *(pline+i)='*';
 *(pline+n+1)='\0';
 printf ("%s", line);
}
```

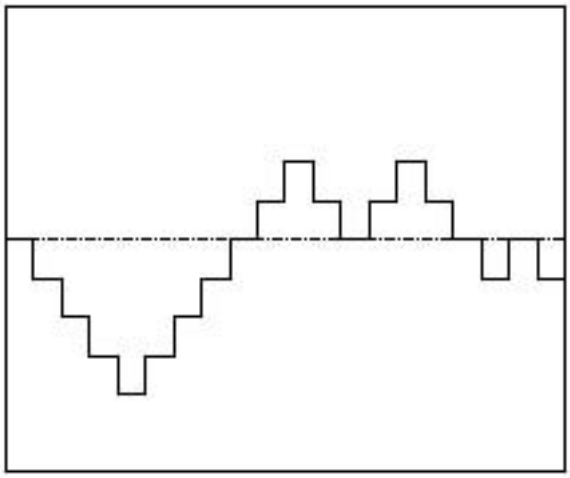
実行結果例は次のようになるはずである.

```
A:¥zwork¥C>rw

Random walk - sum of random sequence of {-1,+1}.
Input seed: 178001
Input sequence length: 15
     -1                                  *0
      0                                   *
     -1                                  *0
      0                                   *
     -1                                  *0
      0                                   *
     -1                                  *0
      0                                   *
     -1                                  *0
      0                                   *
      1                                   0*
      2                                   0 *
      1                                   0*
      0                                   *
      1                                   0*
```

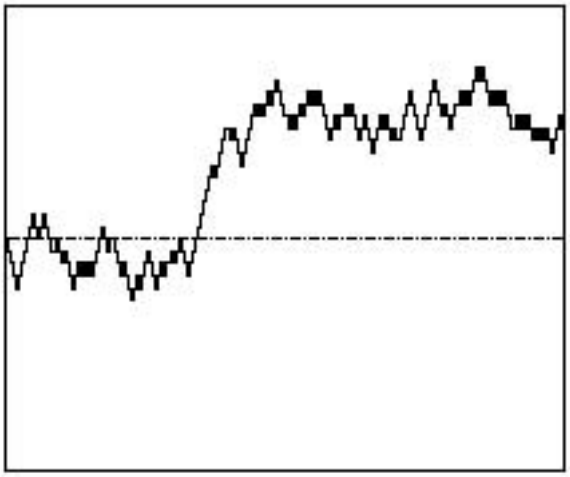
小さい $n$ では殆ど 0 の近くを行ったり来たりするだけで，あまり顕著な傾向が出ないうちに画面の下の方まで来てはみ出してしまう．ファイルに書き出して WZ で眺めるのは良い方法だが，それでもこのプログラムでは 1000 歩ほど進むと左右に星がはみ出てしまう.（ランダムウォークは $n$ の値や乱数の個数などという無粋な言葉を使わず $n$ 歩と数える．酔っぱらいがふらふらしているイメージ.）この程度だといかにもでたらめな数を順に足しているだけというイメージが拭えない.

だが，もっともっと進むと次第に神秘的？な図形が現れる．本格的に神秘的な図形を書くには数値をファイルに書き出して，mathematica と呼ばれる数学処理ソフトか gnuplot と呼ばれる作図ソフトで処理するのがよいと思う.

次の図はランダムウォークの数値データをファイルに保存し， gnuplot で作図させ，結果を図形記述言語 postscript でファイルに保存し，Windos95 上の postscript 言語読みとりソフト ghostview で表示して，コピー機能とペイントのペースト機能でペイントに転送し，bmp ファイルとして保存したものを LaTeX に取り込んで表示した.



`rw.c` では歩数方向が縦方向だったのに対してこの図では横方向になっていることと，星印の代わりに階段状の段々で一歩を表している点が違うが，本質的に `rw.c` の出力と変わらないことは納得できるだろう．歩数を増やすと進んだ距離も自然大きくなってくる．そのままでは紙に収まらなくなるので，次の左図は歩数（横）方向を 1/10 倍，位置（縦）方向を $1/\sqrt{10} \approx 1/3$ 倍，にそれぞれ縮めた.（左端の下向きの V 字型の部分が先上の図形と左図で似ていることに注意！）よく見ればまだ階段状になっていることが分かるが，ぱっと見には階段と言うより複雑な折れ線というイメージになっている．右図は横方向を 1/10 万倍，位置（縦）方向を $1/\sqrt{10\text{ 万}} \approx 1/300$ 倍，にそれぞれ縮めた．独特の神秘的な線画（厳密には線画とも呼べない）が現れる．神秘的な図形が現われるということは，でたらめに並んだ数列を足していくだけで，非常に深い意味での「規則性」が現われるということである.「でたらめ」や「規則性がない」という言葉を軽く見ることはできない．以下の節で，でたらめ（乱数）の中の規則性についてもう少し掘り下げる．



横と縦を $1/N$ 倍と $1/\sqrt{N}$ 倍にして $N$ を大きくしていくのが面白いということだが，このヒントは V.6.4 節で少し触れる．

### V.6.3　乱数の平均値　＝　ランダムウォークの大数の法則

V.5.2 節 の (V.5.2) で乱数の平均

$$E_n = \frac{1}{n}(x_1 + x_2 + \cdots + x_n) = \frac{1}{n}\sum_{i=1}^{n} x_i$$

を考えた．そこでは $[0,1)$ 一様乱数を考えたが，$\pm 1$ 乱数でも全く同様に平均値を考えることができる．これを V.6.2 節 のランダムウォーク $S_n = x_1 + x_2 + \cdots + x_n$ の視点で見ると $E_n = S_n/n$ となる．V.5.2 節 の議論と同様にして $x_1, x_2, \cdots,$ が $\pm 1$ を等しい確率でとる乱数ならば

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} f(x_i) = \frac{1}{2}(f(1) + f(-1))$$

が任意の関数 $f$ に対して確率 1 で成り立つことが分かる.（$\pm 1$ 乱数の場合は任意の関数が可測関数になる.）従って特に,

$$\lim_{n\to\infty} \frac{1}{n} S_n = \frac{1}{2}(1 + (-1)) = 0\,. \tag{V.6.6}$$

言い換えると，$n$ 歩のランダムウォークは必ず $n$ に比べてゆっくりとしか広がらない．まっすぐ右に進めば $(x_1 = x_2 = \cdots = 1)$ $n$ 歩で距離 $n$ かせぐのだから，その一割どころか（1 パーセントどころか 1ppm どころか），全然距離をかせがない，というのは奇妙な感じもする．

この結果はいかめしく言うと大数の (強) 法則 ((Strong) law of large number; LLN) という定理の特別な場合である．大数の法則自体もいろんな条件の下で成り立つが，一番条件の簡単な形で書いておく．

**大数の法則：** 期待値が 0 の独立同分布確率変数列の平均は確率 1 で 0 に収束する．

独立同分布確率変数列という言葉はまだ分からないかも知れないが，理想的な乱数は独立同分布確率変数列の「典型的な」サンプルになっている，というのが,「十分でたらめな数列」という気持ちの一つの言い換えである.（乱数列の言葉は well-defined ではない．しかし，独立同分布確率変数列は well-defined な数学概念であり，大数の法則も証明された定理である.）

大数の法則を図示するプログラムを書いてみよう．今までのプログラム例に慣れていれば，簡単なはずである．次の例では主関数 `main` 以外の関数は全て今まで例示したプログラムに出てくる.（最初から後々のことを意識するのは私には難しいが，特に C 言語のプログラミングでは，後で広く使うのに適した普遍性のある関数を多数用意して，それを組み合わせて段々複雑なプログラムを作っていく，というやり方が正しい.）用いる変数も既出のものばかりで説明はいらないだろう．

```
/* lln.c (Random walk scaled by 1/n  =  average of random numbers) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define length   50
int  irnd     (void);
void rndinit  (int *iteration);
void memori   (void);
void plotstar (int n, int i);

void main (void)
{
 int n, i;  double mean;
 printf ("\nAverages of random sequence of -1, +1. \n");
 rndinit (&n);
 memori ();
 for (mean =0.0, i=1;i<=n;i++)
  {
   mean+= ((double) irnd () - mean)/((double) i);
   printf ("%5.3f     ", mean);
   plotstar (length, floor(0.5+(mean+1.0)/2.0*length) );
   putchar ('\n');
  }
 putchar ('\n');
}

void rndinit (int *iteration) {  random.c に同じ  }
int irnd (void)               {  rw.c に同じ  }
void memori (void)            {  graph.c に同じ  }
void plotstar (int n, int i)  {  graph.c に同じ  }
```

図示すると，ゼロに近づく傾向が見た目で分かるので印象的だが，歩数が少なすぎて，本当にゼロに近づくかはっきりしない．もっと $n$ の大きいところを調べるために途中結果の一部のみ数字のみで出力する．次の例では途中結果を $\texttt{exponent}^n$, $n = 1, 2, 3, \cdots$, のところで書き出していることに注意 ($\texttt{exponent}= 10$)．このような取り方は減少（増大）傾向が $b \times n^{-a}$ に近い場合に，減少の指数 $a$ のおおよその値（数値計算値）を数値計算の結果から読みとるのが容易という意味で，特に有効である．どのようにして読みとればよいかは宿題.

```
/* lln2.c (Random walk scaled by 1/n  =  average of random numbers) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define exponent 10
#define lines    15
int  irnd     (void);
void rndinit  (int *iteration);

void main (void)
{
 int gyo=1, i=0, m=1, n;  double mean=0.0;
 printf ("\nAverages of random sequence of -1, +1. \n");
 rndinit (&n);
 printf ("%15s%17s\n", "n", "mean");
 while (gyo<=lines && i<=n)
  {
   m*=exponent;
   while(i<=m && i<=n)
    {
     mean+= ((double) (irnd ())-mean)/(double) (i+1);
     i++;
    }
   printf ("%17d    %16.14f\n",i-1,fabs(mean));
   gyo++;
  }
}

void rndinit (int *iteration) {  random.c に同じ  }
int irnd (void)               {  rw.c に同じ  }
```

### V.6.4　ランダムウォークのスケーリング　＝　中心極限定理

大数の法則 V.6.3 節 (V.6.6) から，ランダムウォーク $S_n$ は $n$ に比べてゆっくりとしか遠くに行かない（絶対値が大きくならない）．しかし，$n$ が増えると共に大きな値を（も）とることはランダムウォークの図示 (V.6.2 節) によって計算機でも見ることができる．そこで自然に $S_n$ （もう少し正確には $|S_n|$）が $n$ とともに $n^\alpha$ くらいの速さで増大していないだろうか，と想像してみる．ここで $\alpha > 0$ は定数であり,「くらいの速さで増大する」とは $\beta > \alpha$ ならば確率 1 で（つまり任意の乱数列に対して） $\lim_{n\to\infty} \frac{1}{n^\beta} S_n = 0$, $\beta < \alpha$ ならば確率 1 で $\lim_{n\to\infty} \frac{1}{n^\beta} S_n = \infty$, となる，という意味である．もちろんそんな著しいことが成り立つかどうか (V.6.6) だけから分かるはずはない．

そこで，そのような $\alpha$ があると仮定して数値計算で $\alpha$ （があるとすればどんな値か）を求めてみよう．

```
/* clt.c (random walk scaled) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define length   50
#define epsilon  1.0e-1
int  irnd     (void);
void rndinit  (int *iteration);
void memorip  (void);
void plotstar (int n, int i);

void main (void)
{
 double alpha, mean, mean_alpha, sum;  int i, ima, j, n;
 printf ("\nScaling for random walk; sum of random {-1,+1}.\n");
 rndinit (&n);
 memorip ();
 for (mean =0.0, mean_alpha=0.0, j=1, i=1;i<=n;i++)
  {
   mean+= ((double) irnd () - mean)/((double) i);
   sum= mean*((double) i);
   if (fabs(sum) > epsilon && i>1)
    {
     alpha= log(fabs(sum))/log((double) i);
     mean_alpha+= (alpha - mean_alpha)/((double) j);
     j++;
     ima= floor(0.5+mean_alpha*length);
     printf ("%5.3f     ", mean_alpha);
     plotstar (length, ima);
    }
   else
    {
     printf ("          ");
     plotstar (length, -1);
    }
     printf ("   %5.3f  %5.3f", alpha, mean);
     putchar ('\n');
  }
 memorip ();
 putchar ('\n');
}
```

```
void rndinit (int *iteration) {  random.c に同じ  }
int irnd (void)               {  rw.c に同じ  }
void plotstar (int n, int i)  {  graph.c に同じ  }

void memorip (void)
{
 int i;
 printf ("平均α    ");
 putchar ('0');
 for (i=1; i<length/2; i++) putchar (' ');
 putchar ('5');
 for (i=1; i<length/2; i++) putchar (' ');
 putchar ('1');
 printf ("     α       Sn/n");
 putchar ('\n');
}
```

このプログラムでは $\frac{1}{n^\alpha}S_n = 1$ 即ち，$\alpha = \log S_n / \log n$ とおいて，その $\alpha$ の平均を計算している．但し，ランダムウォークが原点に戻る ($S_n = 0$) とその対数は計算できないから困る．このプログラムでは強引に，そういう $n$ に対応する $\alpha$ は計算しないことにしている．この処理はきちんと正当化するのは難しい（そもそも $\alpha$ の平均は数学的には調べづらい量である）が，おおよその傾向をつかむ安直な方法と考えればよい．比較的小さな $n$ でも $\alpha$ が 1/2 近く，ということが見えると思う．V.6.3 節 の `lln2.c` のように，図示を諦め，大きな $n$ の計算を行うようプログラムを書き換えることは宿題としよう．

問題になっている性質を持つ定数 $\alpha$ は存在し，1/2 であることが知られている．中心極限定理 (Central limit theorem; CLT) はもう少し精密な形で与えられている．条件の一番簡単な形で書くと次のようになる．

**中心極限定理：** 平均 0，分散 1 の独立同分布確率変数列の $n$ 項の和 $S_n$ に対して $S_n/\sqrt{n}$ の分布は標準正規分布に弱収束する．

再び分からない言葉が多いと思うが，±1 乱数列に即して言えば，V.5.2 節 の平均と分散（の極限）はそれぞれ $(1-1)/2 = 0$ と $((1-0)^2 + (-1-0)^2)/2 = 1$ であり，これが中心極限定理の平均と分散に対応するので定理がそのまま使える．すると定理の言うことは，ランダムウォーク $S_n$ に対して歩数 $n$ が十分大きければ（いろんな乱数初期値にのうちで） $a < S_n/\sqrt{n} < b$ となる割合はほぼ $\int_a^b e^{-x^2/2}\frac{dx}{\sqrt{2\pi}}$ に等しくなる（$n \to \infty$ で一致する）ということである．サンプルについて確率 1 で成立する大数の強法則と違って，中心極限定理は分布に関する定理である，つまり，いろんな初期値に対する割合（分布）についての定理である．$S_n$ が $n$ とともに大きくなる度合いは，分布の意味で $n^{1/2}$ 程度であることを主張している．(分布に関する定理だから，本当にチェックしようとしたら度数分布を調べる必要がある．`clt.c` はそこで無理がある．無理のない形の定理は V.6.5 節 を参照.)

中心極限定理を用いれば，V.6.3 節 の `lln2.c` で平均値 $S_n/n$ が 0 に近づく速さを議論できる．初期条件に関する割合を $P$ と書くことにすると，中心極限定理から

$$P(\frac{a}{\sqrt{n}} \le \frac{1}{n}S_n \le \frac{b}{\sqrt{n}}) \approx \int_a^b e^{-x^2/2}\frac{dx}{\sqrt{2\pi}}$$

となる．右辺が十分 1 に近くなるように $a, b$ を決めると，かなり多くの初期値に対して $n$ が十分大きければ

$$\frac{a}{\sqrt{n}} \le \frac{1}{n}S_n \le \frac{b}{\sqrt{n}}$$

となる．これが乱数列の平均値 $\frac{1}{n}S_n$ が 0 に向かう速さ（$n$ のどういうべきに従うか）を与える．

### V.6.5　重複対数の法則

V.6.4 節 の中心極限定理は分布に関する定理であったが，ランダムウォーク $S_n$ について $n^{1/2} \times (\log\log 2)^{1/2}$ という増大の仕方を示す定理であって，大数の法則 V.6.3 節 のように個々の数列（サンプル）に対して確率 1 で成立する内容の定理もある．有名なのが重複対数の法則 (Law of iterated logarithm; LIL) である．

**重複対数の法則：** 分散 1 の分布に従う有界な独立同分布確率変数列の $n$ 項の和 $\displaystyle\sum_{k=1}^{n} X_n$ は確率 1 で

$$\limsup_{n\to\infty} \frac{1}{\sqrt{2n\log\log n}} \sum_{k=1}^{n} X_n = 1$$

を満たす．

名前は $\log\log 2$ が対数の対数という形をしているところからきているが，$\log\log x$ という関数は非常にゆっくりとしか変化しない．したがって，確率変数の和の変化の様子を表わしている主要な部分は中心極限定理と同様 $n^{1/2}$ である．

重複対数の法則を数値計算で確かめるのは**極めて難しい**というのが私の第一感である．それは $\limsup$ とは何かを考えるときに困るからである．$\lim_{n\to\infty} a_n = 1$ を数値計算で確かめたければ数列 $a_n$ を $n = 1, 2, \cdots,$ に対して計算して，次第に 1 に近づくことを見ればよい．もし近づき方が遅ければ $a_n$ を $n = 1, 10, 100, 1000, \cdots,$ に対して計算して変化を眺めればよい（以下同様）．しかし，$\limsup_{n\to\infty} a_n = \lim_{N\to\infty} \sup_{n\geq N} a_n$ なので，これが 1 に等しいことを数値計算で確かめたければ $b_N = \sup_{n\geq N} a_n$ を $N = 1, 2, 3, \cdots,$ あるいは $N = 1, 10, 100, \cdots,$ に対して計算して変化を見ないといけない．ところが例えば $b_{10} = \sup_{n\geq 10} a_n$ は第 10 項目以降の無数の $a_n$ の中で一番大きいものであるから，一つの $N$ 毎に無限個の $a_n$ を比較しないといけない．計算機で調べるときは，十分大きな $M$ を持ってきて $\sup_{M\geq n\geq 10} a_n$ を計算し，例えば $M = 10^4, 10^5, 10^6, \cdots,$ と増やしたときにこの値がどういう値に近づくかを推測してやっと $b_{10}$ が一つ決まることになる．これは大数の法則 V.6.3 節 の検証 `lln2.c` に比べて大変難しいということが分かると思う．

参考までに，上の考え方（アルゴリズム）に沿って書いたプログラムを書いておく．

```
/* lil.c (Random walk scaled by sqrt{2n log log n}) */
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define exponent 4
#define lines    16
#define n        100000000
#define seed     178001
int  irnd     (void);

void main (void)
{
 int kou, line, i;
 double scaled, sum=0.0, sup[lines]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,};
 printf ("\nLaw of iterated logarithms for random sequence of -1, +1. \n");
 printf ("limsup_{N}=sup_{M>=n>=N}; M=%10d\n", n);
 printf ("%10s%30s\n", "N", "limsup_N |Sn|/sqrt(2n loglog n)\n");
 srand (seed);
 sum+=(double) irnd (); sum+=(double) irnd (); i=3;
 while(i<=n)
  {
   sum+=(double) irnd ();
   scaled=fabs(sum)/(sqrt(2.0* (double) i * log(log((double) i))));
   kou=exponent; line=1;
   while (kou<=i && line<lines)
    {
     if(sup[line]<scaled) sup[line]=scaled;
     kou=kou*exponent; line=line+1;
    }
   i++;
  }
 kou=exponent; line=1;
 while (kou<=n && line<lines)
  {
   printf ("%17d    %16.14f\n",kou,sup[line]);
   kou=kou*exponent; line++;
  }
}

int irnd (void) {  rw.c に同じ  }
```

私の使っている計算機で無理なくできるのは次の結果までだった．結果から見て，プログラムを工夫するだけでは限界があり，もっと数学を工夫しないといけない（計算機で調べるにしてももっと収束の速い量に直して計算すべきである，という意味）と思うが何如であろうか．この先は諸君に工夫して頂きたいと思う．それにしても驚くことは，このように計算機で見いだしにくい結果がきちんと数学的に証明できるという事実である．(逆

に，既に証明がある以上は，それに沿ったアルゴリズムを工夫すればもっと良質の結果が出るであろう．もちろん，本当の数値計算としてはそれは順序が逆であって，まだ数学的に解決していない問題に対して良い結果を得るようにアルゴリズムを工夫していくことで結果的に何が起こっているかを知り，数学的な証明がアルゴリズムから見えてくるのが理想である.）

```
Law of iterated logarithms for random sequence of -1, +1.
limsup_{N}=sup_{M>=n>=N}; M= 100000000
         N  limsup_N |Sn|/sqrt(2n loglog n)

               4    1.15115935070832
              16    1.15115935070832
              64    1.15115935070832
             256    1.15115935070832
            1024    1.15115935070832
            4096    1.07854439782295
           16384    0.96496142145575
           65536    0.96496142145575
          262144    0.74146864133015
         1048576    0.74146864133015
         4194304    0.74146864133015
        16777216    0.74146864133015
        67108864    0.74146864133015

Law of iterated logarithms for random sequence of -1, +1.
limsup_{N}=sup_{M>=n>=N}; M= 200000000
         N  limsup_N |Sn|/sqrt(2n loglog n)

               4    1.15115935070832
              16    1.15115935070832
              64    1.15115935070832
             256    1.15115935070832
            1024    1.15115935070832
            4096    1.07854439782295
           16384    0.96496142145575
           65536    0.96496142145575
          262144    0.75607929936224
         1048576    0.75607929936224
         4194304    0.75607929936224
        16777216    0.75607929936224
        67108864    0.75607929936224
```

## V.7　その少し先の数学

### V.7.1　大偏差値原理

V.6.4 節では $n$ 歩のランダムウォーク $S_n$ はほぼ $\sqrt{n}$ 程度の広がりの中にいることを見た．特に，V.6.3 節で $\frac{1}{n}S_n$ はほぼ 0 であることも見た．従って，$S_n \geq an$ ($a$ は正定数) となることは ($n$ が大きければ）めったに起

きない．しかし，$n$ が有限である限り少しは起きる（いくつかの不運な，あるいは幸運な，初期条件 seed に対してある大きな $n$ で $S_n \geq an$ となる，ということ）．そこで「どれくらいそういうことが少ないか」が問題になる．

これは単に趣味的な問題と言うだけではなく，中心極限定理からのずれが小さいということを証明することによって，いろんな量が中心極限定理の示唆するとおりになることを証明する補題の役割を果たす．一般にもっと複雑な問題に至るまでこの種の評価がどうあるべきか，というガイドラインが知られていて，それを大偏差値原理 (Large deviation principle; LDP) と呼ぶ．そういう複雑な問題では物理の熱力学で出てくる「エネルギーとエントロピーのバランス」という概念がちょうど大偏差値原理に対応する（というより，偉い数学者たちがそういう視点からこれらのガイドラインを確立したようにみえる）．

ここに出てくる程度の問題を大偏差値原理と呼ぶのはおこがましいかも知れないが，用語の紹介を兼ねてそう呼ばせていただく．ランダムウォーク $S_n = x_1 + x_2 + \cdots + x_n$ については，次のことが成り立つ．

$1 > x > 0$ を定数とする． $S_n \geq nx$ となる確率（乱数列では，これが成り立つ初期条件の，全初期条件に対する割合に対応する）を $P(S_n \geq nx)$ と書くことにすると，

$$\lim_{n\to\infty} \frac{1}{n} \log P(S_n \geq nx) = \frac{1}{2} \log \frac{1}{1-x^2} - \frac{x}{2} \log \frac{1+x}{1-x} \tag{V.7.7}$$

が成り立つ．

十分大きな $n$ に対して $P(S_n \geq nx)$ は $n$ とともに指数関数的に減衰することを意味している．その指数が右辺のように $x$ の関数としてきちんと計算できるところも興味深い．指数関数的減衰は速いので，その傾向を確かめるだけならば比較的やさしい数値計算のはずであるが，分布に関する性質なので，中心極限定理 (V.6.4 節) と類似の意味で数値計算は難しい．

### V.7.2 拡散方程式

ランダムウォーク $S_n$ については（言い換えれば,「理想的」な $\pm 1$ 乱数の和ならば） $S_{n+1} = S_n \pm 1$ となる確率（初期条件の割合）がそれぞれ $\frac{1}{2}$ である．前節のように確率（割合）を $P$ で表すことにすれば $n+1$ 歩目に $z$ にいる確率 $P(S_{n+1} = z)$ は，$n$ 歩目に $z-1$ にいて次の一歩で確率 $\frac{1}{2}$ で右に進む場合と，$n$ 歩目に $z+1$ にいて次の一歩で確率 $\frac{1}{2}$ で左に進む場合の和である．よって

$$P(S_{n+1} = z) = \frac{1}{2} P(S_n = z-1) + \frac{1}{2} P(S_n = z+1)\,,\ n = 0, 1, 2, \cdots,\ x \in \{0, \pm 1, \pm 2, \cdots\}\,.$$

$u(n, z) = P(S_n = z)$ とおくと，$u$ は $0 \leq u \leq 1$ に値をとる 2 変数実数値関数であって，

$$u(n+1, z) - u(n, z) = u(n, z+1) - 2u(n, z) + u(n, z-1) \tag{V.7.8}$$

を満たすことが分かる．さて，$\epsilon > 0$ に対して

$$f_\epsilon(t, x) = u([\frac{t}{\epsilon}], [\frac{x}{\sqrt{\epsilon}}])$$

とおく．ここで $[\cdot]$ は $\cdot$ を超えない最大の整数．(V.7.8) の両辺を $\epsilon$ で割って $\epsilon \to 0$ とすると,（$f_\epsilon$ が十分なめらかな関数 $f$ に収束するならば）

$$\frac{\partial f}{\partial t}(t, x) = \frac{\partial^2 f}{\partial x^2}(t, x) \tag{V.7.9}$$

となる．この方程式は拡散方程式と呼ばれる偏微分方程式である．たいへん基本的な方程式で，数学のあらゆる分野に現れるだけでなく，応用上も非常に多くの分野に現れる．

ランダムウォーク（のある極限）がこのように基本的な偏微分方程式に関連していることはこの先に大きな数学の分野があることを予感させる．これは確率解析の一つの入り口である．

## V.8 実習およびレポート提出作成の手順

### V.8.1 プログラミングの作業の要約

(1) プログラム (C 言語) やレポート (LaTeX) のファイルの打ち込み (WZ エディタ)
(2) プログラムのコンパイルや実行 (Cygnus, UNIX モード)

以下は，木田プリントの準備のページからの抜粋．詳しくは木田プリント参照．

(1) WZ というアイコンのダブルクリック．保存は c:\cygnus\temp というディレクトリ.（続けてフロッピーディスク a:\ に保存してもよい.）
(2) Cygnus というアイコンのダブルクリック.
  (a) ls でファイルの一覧．cp ファイル名.c a: でフロッピーディスクにファイルコピー.
  (b) gcc ファイル名.c -o ファイル名.exe でコンパイル後に ./ファイル名.exe でプログラム実行.

なお，プリントのプログラムを gcc にかけると,「main なのに int 型でない」といったような Warning(注意) が (英語で) 出るかもしれないが，この Warning **のみ**無視してよい.

### V.8.2 乱数列の記録

手軽な順に例えば以下のような方法がある.

**手で画面を写す.** 手軽だが，書き間違いがあり，また，長い乱数列を記録するのは退屈で時間もかかる．勧められない.

**画面のハードコピーをとる.** 実行して結果が出たところで Alt を押しながら print screen のキー（それぞれキーボードの左下と右上にある）を押す（これでアクティヴな窓のハードコピーがペーストバッファーにコピーされる）．次にデスクトップ左下隅にマウスを移動し，スタートボタンをクリックして，メニューから [プログラム] [アクセサリ] [ペイント] の順に選び，ctrl を押しながら文字 v （両方キーボードの左手の範囲にある）を押す（サイズを大きくしてもいいかと聞いてくることがあるが，肯定的に答える）．あとはペイントソフトのコマンドを利用して，必要な範囲を切り取ったり白黒反転したりして編集してもよい．ペイントの詳しい使い方はメニューの [ヘルプ] [トピックの検索] を見よ．窓の中の上のほうのメニューバーから ファイル をクリックして [印刷] を選べば印刷できるし， [名前を付けて保存] を選べば bmp ファイルとして保存できる．サイズが大きくなるので，保存の際は [ファイルの種類] を 16 色ビットマップに選ぶこと.

LaTeX に絵として取り込むことができて，計算機室の dvi 処理ソフトなら絵を文書に埋め込んだ状態で印刷できる．V.8.3 節 を参照．このプリントの出力例はこのようにして作成した.

手軽だが，ファイルは大きくなり，画面からはみ出ると記録できない．最初は画面で確かめるだけのつもりだったが，結果を見たら残しておきたくなった，というときに便利.

なお，この方式は web browser の表示で不便なので，レポートでは勧めない.

**リダイレクト.** ./random.exe として実行する代わりに./random.exe >> random.dat として実行する．プログラムに書き込んだ入力を促す表示 (Input seed など) が表示されなくなるが，構わず seed の値と出力する乱数の個数を入力するとプログラムは動く．実行が終わるとプロンプト（画面の次の行に出るおきまりの文字列）が出るので，ls と打ち込むと random.dat というファイルができていることが分かる．cat random.dat と打ち込んでみると，このファイルに数字が書き込まれていることが分かる．cp random.dat a: としてフロッピーディスクに保存しておくこと．エディタ (WZ) で開いて編集することもできる．例えば，レポート report1.tex も WZ で同時に開いておいて，random.dat からコピーとペーストで結果を report1.tex

にコピーすれば，そのままレポートのデータとなる．慣れれば手で書き写すよりはるかに正確で速く便利である．

（注：コピーとペースト：コピーしたい範囲の先頭にマウスを移動してマウスでドラッグ (左ボタンを押したまま動かすこと) で範囲の最後にマウスを移動することでコピーしたい範囲の白黒を反転させる，ctrl を押しながら文字 c を押す．report1.tex の窓に移って，コピーしたい場所でマウスを左クリックしてから ctrl を押しながら v を押すとペーストされる．）

問題点は，入力を促す表示が（ファイルに書かれて）画面には出ないので入力しにくい点である．解決にはプログラムを手直しする必要があるが，プログラムを手直しするなら次項のようにファイルに書き出すプログラムに書き換えるのがよい．

早く終わるが結果は多量に出るプログラムの時で，最初は画面で確かめるだけのつもりだったが，結果を見たら残しておきたくなった，というときに便利．

**プログラムを手直ししてファイルに書き出す．**（木田プリント X–29 ページ以降参照）．例えば V.4.1 節 の random.c の場合，main 関数の最初に

```
FILE *fp;
```

という行を入れて，さらに

```
for (i=1;i<=n;i++) printf ("%15.12f\n", rnd ());
```

を

```
fp=fopen("random.dat","a");
for (i=1;i<=n;i++) fprintf (fp,"%15.12f\n", rnd ());
fclose(fp);
```

に書き換えればよい．長い出力を記録したいときはもっとも正当な方法．

出力方法の大雑把な使い分け．

**画面表示：** printf．プログラムのテストのための短い出力．

**ファイル出力：** fopen, fprintf．記録に残したい結果や他のソフトやプログラムを用いて詳しく調べたい多量のデータ．

煩雑で長くなるので，このプリントでは本文では出力方法の使い分けの注意を繰り返さない．各自で目的に応じて以上のような書き直しを行ってプログラムを実行することを前提にして例示していることに注意．

### V.8.3 LaTeX によるレポート作成

LaTeX による文書作成は佐藤プリントを復習すること．
以下は文書作成の流れについての佐藤プリントからの抜粋要約である．

**文書ファイルの作成：** エディター（WZ）を用いて文書ファイルを作り，フロッピーディスク A:\ を選んで report1.tex という名前で保存する．

**TEX による組版：** report1.tex を TEX コンパイラ（virtex）でコンパイルする．コンピュータはファイル report1.dvi を作成する．

**印刷・画面表示ソフトによる出力：** report1.dvi をプレビューア（dviout）にかけると，文書をモニター画面上でプレビューしたり印刷することができる．

WZ はデスクトップ（画面）上にあるアイコンをダブルクリックすれば起動する．LATEX ファイルは佐藤先生担当分で作ったものがフロッピーディスクに残っているだろう．既存のファイルを使って書き換えるほうが一から作るより楽．WZ の窓の上部メニューバーから [ファイル] [開く] を選び，適当な（短い）LATEX ファイル A:****.tex を開く．（ファイルの種類の欄は [通常] にする．）開いたら元のファイルを壊さないよう，名前を変える．即ち，[ファイル] [名前を付けて保存] を選んで A:\reort1.tex としてフロッピーディスクに保存する．あとは，レポートの解答を編集して保存する．

文書作成については佐藤プリントにたいへん懇切丁寧に説明してあるから，よく復習しておくこと．以下，佐藤プリントを読んでいることを前提にして，簡単に report1.tex の編集例を説明する．なお，佐藤プリントに注意してあるとおり，\ は ¥のこと．

```
\documentclass[a4paper]{jarticle}
\begin{document}
第１回レポート課題

yca078z 服部哲弥

[1] \verb|xxxxxxx|

[2]
seed=1xx001
\begin{verbatim}
0.xxxxxx
0.xxxxxx
 ...
0.xxxxxx
\end {verbatim}

[3]
seed=3xx001
\begin{verbatim}
0.xxxxxx
0.xxxxxx
0.xxxxxx
0.xxxxxx
0.xxxxxx
\end {verbatim}

\end{document}
```

**\end {verbatim} の行， end の次に１文字空いているように見えるかも知れないが，空けてはいけない！**

xxxxxx のところに解答を入れてファイルを作り，例えば report1.tex という名前で保存する．問 [2]，問 [3] の xxxxxx のところは random.dat から WZ のコピーとペーストで持ってくればよい．用いた seed の値も必ず書くこと．

report1.tex を TEX コンパイラ（virtex）でコンパイルすればレポートができる．virtex はデスクトップ左下にマウスを移動して，スタートボタンを左クリックし，[プログラム] [p-TeX2.1.5] を順に選ぶと入っている．窓の上部のメニューバーから [ファイル] [TEX で処理する] を選び，編集した TEX ファイル名 (A:\ にある report1) を選ぶか A:\report1 と書き込めば処理が始まる．

エラー表示が出て止まったら x キーを押してから Enter キーを押して処理を中断する．WZ の窓に移って修正を繰り返す．コンパイルが終了し report.dvi ファイルができあがったならば, メニューから [ファイル] [プレビュー] を選べばプレビューア dviout が起動する．dviout で表示すると出力は次のようになる．

第１回レポート課題
yca078z 服部哲弥
[1] xxxxxxx
[2] seed=1xx001

0.xxxxxx
0.xxxxxx
 ...
0.xxxxxx

[3] seed=3xx001

0.xxxxxx
0.xxxxxx
0.xxxxxx
0.xxxxxx
0.xxxxxx

上記はレポート作成の考え方を示す例であって，レポートとしては**悪いレポート**である．単に数字だけでなく説明を簡潔・十分に書かなければいけない．さらに気がついたことや，問を自分で発展させて検討した結果も書くべきである．また，TEX の文書整形命令を最小限しか使っていない点でも TEX の使い方の未熟なレポートである．佐藤プリントで学んだ文書整形命令を駆使し，プレビューアで確認しながら WZ でさらに修正を繰り返すことで，見やすいレポートを書くべきである．

レポートが完成したら，V.8.4 節 に従って report1.tex と report1.dvi を各自の web page のディレクトリに転送する（以後のレポート提出作業は V.8.4 節 を参照）．

画面のハードコピーを作って，ペイントソフトで問 [2]，[3] の答をそれぞれ random2.bmp と random3.bmp に保存した場合は次のようなファイルを report1.tex とする．

```
\documentclass[a4paper]{jarticle}
\begin{document}
[1] \verb|xxxxxxx|

[2]
seed=1xx001

\begin{minipage}{7.6cm}
\makebox[7.60cm][l]{\special{bmpfile=random2 hsize=7.60cm vsize=12.63cm}}
\vspace*{12.63cm}
\end{minipage}

[3]
seed=3xx001

\begin{minipage}{7.6cm}
\makebox[7.60cm][l]{\special{bmpfile=random3 hsize=7.60cm vsize=12.63cm}}
\vspace*{12.63cm}
\end{minipage}

\end{document}
```

hsize=7.60cm などの数字はハードコピーの横幅，vsize=12.63cm は縦幅，を表す．これらの数値はペイントソフトの [変形] [図形の色とサイズ] のメニューを選べば知ることができる．

**この方法でレポートを作った場合は，bmp ファイルも web site に転送し，そのファイル名を電子メールで服部まで連絡すること．(Web browser では恐らく bmp 部分は確認できないので，別途処理する必要がある．)**

### V.8.4 レポート提出方法

成績はプリントの最後にある 3 回のレポートを提出することによる．**レポート提出は次の手順による．**

(1) LaTeX によるレポート文書 (例えば report1.tex) とそれをコンパイルした dvi ファイル (例えば report1.dvi) を自分のフロッピーディスクに保存する．(この段階までは佐藤先生担当分のプリントと V.8.3 節 参照．dvi ファイルは virtex で tex ファイルをコンパイルした時点で自動的に保存される．)

(2) 自分のホームページディレクトリにこれらのファイルを FTP ソフト (WS_FTP など) で転送（アップロード）する．内田さんが FTP ソフトのアイコン（絵のボタン）をデスクトップ（画面）に用意してくれているはず．それをダブルクリックすればよい．Host name (address) は fx124 または 150.93.96.124，ID は各自の学籍番号 (yca0***)，password は 1 年生のとき木田先生の授業で各自変更したはず．忘れた人は内田さんに問い合わせる．接続 (connect) かセッション (session) など（ソフトによって違う）のボタンを選ぶか窓が開いているところでこれら (host, ID, password) をしかるべきところに記入すれば自分のホームページディレクトリにファイルを転送できるようになるはず．
転送元はフロッピーディスク (A:\) を選ぶ．WS_FTP というソフトの場合は，窓の左の欄を下のほうにスクロールすると [-A-] が現れるので，それを選んで ChgDir を押せば，report1.tex, report1.dvi 等が見えてくるはず．ソフトが違っても作業は似ている．右欄も public_html を選んで ChgDir を押さないといけないかもしれない（1 年の木田先生の授業で習ったとおりにするのが正しい）．

LaTeX ファイルは ascii mode で転送し，dvi ファイルは binary mode で転送する．(WS_FTP では窓の下の方にクリックして選ぶ場所がある．転送したいファイルを左の欄から選びファイル名をクリックすると色が反転する．中程の右矢印を押せば転送する.)

転送先のファイル名を（大文字小文字の区別も含めて）転送先ディレクトリで確認し，記録しておくこと．

(3) ブラウザ (Netscape など) で転送したファイルが開くことを確認する．ブラウザも内田さんがアイコンをデスクトップに用意してくれているはず．それをダブルクリックすればよい．上部メニューバー近く [ジャンプ] または [場所] となっている空欄をマウスでクリックし，

`http://fx124.rikkyo.ac.jp/~yca***/report1.dvi` または

`http://150.93.96.124/~yca***/report1.dvi` などと書いて Enter キーを押すと転送したレポートが見えるはず．(*** のところは各自の学籍番号が入る．ファイル名は大文字小文字の区別にも注意.)「セキュリティに問題があります」というようなメッセージが出るかもしれないが，[開く] を選んで [OK] を押せば dviout が起動してファイルを見ることができる．

(4) ブラウザで服部のホームページ

`http://150.93.96.124/math/hattori/hattori.htm` の下の　／講義のページ／計算機応用のページ　を開き，そこにある投稿用のフォームに従って，転送したレポート文書の URL (＝文書の住所＝転送先計算機名＋ファイル名) を，

`http://150.93.96.124/~yca078z/report1.tex` と

`http://150.93.96.124/~yca078z/report1.dvi` のように書き込んで，投稿ボタンを押す．yca078z の部分は各自のホームページのディレクトリ名，report1 の部分は転送したレポート文書のファイル名（大文字小文字の区別にも注意）.

正しく行えば画面が「Thank you very much for the report.」となる.「Back to root page.」をクリックして元に戻り，投稿ボタンの下のリンクからレポート提出状況のページを開いてリストに自分の名前が追加されたことを確認し，先の項目と同様にそこに貼ってあるリンク (report1.dvi) をクリックして自分のレポートを見ることが出来ることを確認する.「File Not Found.」などが出た場合は投稿フォームの記入が間違っていた可能性があるからやり直す.（ＢＢＳのリストに載っているのにリンクからファイルを開けない場合は連絡せよ.）

(5) 問題があると画面が「Error: ...」となる.「Error: Complete ...」か「Error: Input ...」の場合はフォームの入力が不完全だったのだから「Back to root page.」をクリックして元に戻り，もう一度上記をやり直す．それ以外の Error は連絡せよ．

(6) 念のためメールソフト (WinYAT，但し一部の計算機のみ) を用いるか，あるいは，テルネットソフト (Tera Term) を用いて７号館計算機センター (ax253) に接続しメールソフト (mnews) を用いるか，などによって服部 hattori@rkmath.rikkyo.ac.jp あてにメールを出して，学籍番号，名前（英数字でなくてよい），何回目のレポートか，レポートのＵＲＬ，を書いた確認用のメールを出すこと.（実習当日の提出の際はメールの代わりに服部への口頭の確認を認める．早期提出の場合は必ず確認のメールを出してほしい.）

19980630–0705;0908–20;23–
30;1021;27;1111;

計算機応用演習

第 V 章担当：服部哲弥

# 第 V 章

## “1 次元ランダムウォーク（数値計算入門）” レポート課題 1

以下の問の答を記した LaTeX ファイルを作成し，V.8.4 節 に指定した方法に従って提出せよ．LaTeX ファイルの作成作業の概説を V.8.3 節 に書いておいた．

[1] V.4.1 節 のプログラム `random.c` の `include` 命令部の空欄を，プログラムが動くように埋めよ．

[2] プログラム `random.c` を `WZ` を用いて作成し，`gcc` でコンパイルし，`./random.exe` によって実行せよ（V.8.1 節 参照）．`seed` は，学籍番号の下 2 桁を `nm` とするとき `1nm001` とせよ．(例：78 番の人は seed= 178001.）得られた乱数列の最初の 10 個の数を報告せよ．
数の記録は手で書き写すよりも計算機で処理することを強く勧める．いくつかの方法を， V.8.2 節 に書いておいたので参考にしてほしい．

[3] `./random.exe` をもう一度実行せよ．`seed` は，学籍番号の下 2 桁を `nm` とするとき `3nm001` とせよ．(例：78 番の人は seed= 378001.）得られた乱数列の最初の 5 個の数を報告せよ．

19980630–0705;0908–20;23–
30;1021;27;1111;

計算機応用演習

第 V 章担当：服部哲弥

# 第 V 章

## “1 次元ランダムウォーク（数値計算入門）” 宿題 1

第 2 回授業日より前に以下の解答を紙に書いておくこと（ファイルに入力しておけばなお良い）．これらはレポート課題 2 の一部になる．

なお，レポートに際して共同研究を認める．但し，レポートに共同研究者の学籍番号と名前を**必ず**明記すること．また，役割分担がはっきりしている場合はその内容も明記すること．

[1] **(i)** $[0,1)$ 一様乱数列データを幅 0.1 の bin に分割して度数分布を調べる．100 万個のデータがあるとき各 bin の度数はそれぞれほぼいくらになるはずか？V.5.1 節 を参照して乱数列が十分長いときの理論値に基づいて答えよ．

**(ii)** $[0,1)$ 一様乱数列データの平均と分散は（乱数列が十分長いとき）どういう値に近づくはずか？理論値を V.5.2 節 に従って求めよ．

[2] **(i)** V.4.2 節 のプログラム `randomb.c` の `rnd` 関数の中の空欄を，同じ節に引用した木田プリントの合同法のアルゴリズムに一致するように埋めよ．

**(ii)** `randomb.c` の中のどれか 1 つの `int` 定義文を `unsigned int` に換えれば正しく $[0,1)$ 一様乱数を生成するようになる．どこを換えれば良いか答えよ．

**(iii)** 空欄を埋め `int` 文を 1 カ所修正した `randomb.c` を V.4.2 節 に従って完成せよ．

[3] **(i)** 平均値を求める 2 つの方法（V.5.2 節 の (V.5.2) と (V.5.5)）が（理論上）同じ値を与える（はずである）ことを証明せよ．

**(ii)** V.5.3 節 で引用した木田・牧野の本の中の桁落ちの例を C 言語のプログラムに書き直せ．

[4] 気づいたことや感想などを自由に記述せよ．

19980630–0705;0908–20;23–
30;1021;27;1111;

計算機応用演習

第 V 章担当：服部哲弥

# 第 V 章

## “1次元ランダムウォーク（数値計算入門）” レポート課題2

以下の問の答を記した LaTeX ファイルを作成し，V.8.4 節 に指定した方法に従って提出せよ．

なお，レポートに際して共同研究を認める．但し，レポートに共同研究者の学籍番号と名前を**必ず**明記すること．また，役割分担がはっきりしている場合はその内容も明記すること．共同研究の場合は学籍番号と名前を連名とし，一人が代表して自分の web site にレポートファイルを単名の場合と同様に転送すること．しかし，ＢＢＳへの投稿は全員が行なうこと.（レポートに名前があってもＢＢＳに投稿していない場合は採点しない.）この場合ファイルのＵＲＬは代表者のディレクトリになることに注意．例えば `yca078z` と `yca000a` が共同研究して `yca000a` が代表で `report2.tex` と `report2.dvi` を自分の web site に転送した場合，二人ともＢＢＳの `URL` 欄は

`http://150.93.96.124/~yca000a/report2.tex` および
`http://150.93.96.124/~yca000a/report2.dvi` になる．

[1] **(i)** $[0,1)$ 一様乱数列データを V.5.1 節 のように幅 0.1 の bin に分割して度数分布を調べる．100 万個のデータがあるとき各 bin の度数はそれぞれほぼいくらになるはずか？乱数列が十分長いときの理論値に基づいて答えよ（宿題１）．

**(ii)** $[0,1)$ 一様乱数列データの平均と分散は（乱数列が十分長いとき）どういう値に近づくか?理論値を V.5.2 節 に従って求めて答えよ（宿題１）．

[2] **(i)** V.4.2 節 のプログラム `randomb.c` の `rnd` 関数の中の空欄を，木田プリントの合同法のアルゴリズムに一致するように埋めよ（宿題１）．

**(ii)** 空欄を埋めた `randomb.c` を V.4.2 節 に従って完成せよ．第１回レポートの `random.c` にならって，ファイルに保存しコンパイルして実行せよ．`seed` は，学籍番号の下２桁を `nm` とするとき `1nm001` とせよ.（例：78 番の人は seed= 178001.）得られた乱数列の最初の 10 個の数を報告し，$[0,1)$ に値をとる一様乱数としてどこが不適切か指摘せよ．

**(iii)** `randomb.c` の中のどれか１つの `int` 定義文を `unsigned int` に換えれば正しく $[0,1)$ 一様乱数を生成するようになる．どこを換えれば良いか答えよ（宿題１）．

[3] 学籍番号下２桁を 3 で割ったとき $n$ 余る人 $(n=0,1,2)$ はそれぞれ次の問のうち $(n)$ 番に答えよ．

(0) V.5.1 節 の `dosu.c` を用い（るか別にプログラムを自作し）て V.5.1 節 の例のように $n=1,100,10000,100$ 万 について乱数列の度数分布を報告せよ．`seed` は，学籍番号の下２桁を `nm` とするとき `4nm001` とせよ.（例：78 番の人は seed= 478001.）上の問題で求めた理論値への近づき方について気づいたことを述べよ．

(1) V.5.2 節 の `bunsan.c` を用い（るか別にプログラムを自作し）て $n=150$ 万個までの乱数の平均値 $E_n$ と分散 $V_n$ の値を $n=10$ 万 毎に報告せよ．`seed` は，学籍番号の下２桁を `nm` とするとき `4nm001` とせよ.（例：78 番の人は seed= 478001.）上の問題で求めた理論値への近づき方について気づいたことを述べよ．

(2) V.5.3 節 の `seido.c` を用い（るか別にプログラムを自作し）て実数変数の有効精度は何桁あるか検討・報告せよ．また，`seido.c` で情報落ちの例も見られるはずであるが，どのような結果を得たか検討・報告せよ．

[4] 次のうちいずれかに答えよ.

(a) 平均値を求める２つの方法（V.5.2 節 の (V.5.2) と (V.5.5)）が（理論上）同じ値を与える（はずである）ことを証明せよ（宿題１）.

(b) V.5.3 節 で引用した木田・牧野の本の中の桁落ちの例を C 言語のプログラムに書き直しプログラムを報告せよ（宿題１）. 但し，本に示唆されているように出力文をループの中に入れること. それをコンパイル・実行し，何が起きたか結果を報告せよ.

(c) 気づいたことや感想などを自由に記述せよ（宿題１）.

19980630–0705;0908–20;23–
30;1021;27;1111;

計算機応用演習

第 V 章担当：服部哲弥

# 第 V 章

# “1 次元ランダムウォーク（数値計算入門）” 宿題 2

第 3 回授業日より前に以下の解答をファイルに入力しておくこと．これらはレポート課題 3 の一部になる．間の休みの週も利用してできるだけ課題 3 もやり抜いてしまうことを薦める．

なお，レポートに際して共同研究を認める．但し，レポートに共同研究者の学籍番号と名前を**必ず**明記すること．また，役割分担がはっきりしている場合はその内容も明記すること．

[1] V.6.2 節 の `rw.c` にならって，V.6.1 節 の `graph.c` を関数 `rnd` の代わりに関数 `irnd` を使って書き直せ．

[2] V.6.3 節 の大数の法則から `lln.c` や `lln2.c` の乱数列の平均 $S_n/n$ が，(正しい乱数ならば) $n$ が大きくなると 0 に近づくことが期待されるが，どのような速さでゼロに近づくか．V.6.4 節 の最後の部分の議論に基づいて $n$ の何乗程度になるか，その指数を求めよ．

[3] **(i)** V.8.2 節を参考にして，V.6.3 節 の `lln2.c` を書き換えて，結果を画面でなくファイル `lln3.dat` に書き込むプログラム `lln3.c` を作れ．

**(ii)** V.6.4 節 の `clt.c` を書き換えて，`lln2.c` のように途中結果を全て書き出さず数カ所だけ出力し，大きな $n$ まで計算するようにし，かつ，V.8.2 節を参考にして，結果をファイル `clt2.c` に書き込むようにしたプログラム `clt.3` を作れ．

[4] 次のうち一つ以上に答えよ．

(a) V.6.3 節 で，$n$ 個の乱数の平均（$n$ 歩のランダムウォーク $S_n$ に対して $S_n/n$）が $b \times n^{-a}$ に近い速さで 0 に収束するとき，減少の指数 $a$ のおおよその値を `lln2.c` の結果から読みとりやすいという趣旨の説明が書いてある．実際に `lln2.c` の実行結果，即ち，$n = 10, 100, 1000, \cdots, 10^k$ における $S_n/n$ の値が与えられたとき，そこから $a$ のおおよその値をどのようにして読みとればよいか，その方法を考案せよ．(出力データ数 $k$ は実際に実行するときの計算時間などから各自判断することになるが，ここでは何かあまり大きくない整数として何を使うことになっても対応できるように考察しておくこと．)

(b) 重複対数の法則によれば，V.6.5 節 の `lil.c` の結果は 1 に近づくべき量を計算しているはずであるが，例示してある結果を見ると，1 より減少する傾向があるように見える．その理由を考えよ．

(c) V.6.5 節 の `lil.c` のような結果から重複対数の法則（右辺が 1 になること）を確認するためには数値計算結果をどう分析すればよいか？

(d) V.6.5 節 の重複対数の法則をランダムウォークの場合に証明するか，証明の載っている文献を探せ．単行本の場合は，著者，題名，発行年，第何版か，発行会社，章節番号，を記録すること．雑誌論文の場合は，著者，題名，発行年，雑誌名，第何巻か，証明のページ範囲，を記録すること．Web page の場合は証明の掲載されている URL (`http://...`) を記録すること．

(e) V.7.1 節 の (V.7.7) を証明するか，証明の載っている文献を探せ．単行本の場合は，著者，題名，発行年，第何版か，発行会社，章節番号，を記録すること．雑誌論文の場合は，著者，題名，発行年，雑誌名，第何巻か，証明のページ範囲，を記録すること．Web page の場合は証明の掲載されている URL (`http://...`) を記録すること．

(f) V.7.2 節 において (V.7.8) から (V.7.9) を導け．

19980630–0705;0908–20;23–
30;1021;27;1111;

計算機応用演習

第 V 章担当：服部哲弥

# 第 V 章

# “1次元ランダムウォーク（数値計算入門）” レポート課題 3

以下の問の答を記した LaTeX ファイルを作成し，V.8.4 節 に指定した方法に従って提出せよ．

なお，レポートに際して共同研究を認める．但し，レポートに共同研究者の学籍番号と名前を**必ず**明記すること．また，役割分担がはっきりしている場合はその内容も明記すること．共同研究の場合は学籍番号と名前を連名とし，一人が代表して自分の web site にレポートファイルを単名の場合と同様に転送すること．しかし，ＢＢＳへの投稿は全員が行なうこと．（レポートに名前があってもＢＢＳに投稿していない場合は採点しない．）この場合ファイルのＵＲＬは代表者のディレクトリになることに注意．例えば `yca078z` と `yca000a` が共同研究して `yca000a` が代表で `report3.tex` と `report3.dvi` を自分の web site に転送した場合，二人ともＢＢＳの URL 欄は
`http://150.93.96.124/~yca000a/report3.tex` および
`http://150.93.96.124/~yca000a/report3.dvi` になる．

[1] 学籍番号下 2 桁を 2 で割ったとき $n$ 余る人 $(n = 0, 1)$ はそれぞれ次の問のうち $(n)$ 番に答えよ．

**(0)** V.6.1 節 の `graph.c` を実行し，結果（数字の列）を 100 個報告せよ．`seed` は，学籍番号の下 2 桁を `nm` とするとき `5nm001` とせよ．（例：78 番の人は seed= 578001.）

**(1)** V.6.2 節 の `rw.c` にならって，V.6.1 節 `graph.c` を `rnd` の代わりに `irnd` を使って書き直せ．書き直したプログラムの `main` 関数の部分だけをを報告せよ（宿題２）．書き直したプログラムを実行して最初の 3 個の値を報告せよ．`seed` は，学籍番号の下 2 桁を `nm` とするとき `5nm001` とせよ．（例：78 番の人は seed= 578001.）

[2] 学籍番号下 2 桁を 2 で割ったとき $n$ 余る人 $(n = 0, 1)$ はそれぞれ次の問のうち $(n)$ 番に答えよ．
いずれの場合もレポートへのプログラムの書き込みは，手で書き写すのでなく，`WZ` でファイル `***.c` を開き，`ctrl+a`（`ctrl` キーを押しながら文字キー `a` を押す）でプログラム全体を選び，`ctrl+c` でコピーし，`WZ` の [ファイル] [開く] で `report3.tex` を開いて，適切な場所にカーソルを動かしてから `ctrl+p` でカーソル位置に張り付けること．V.8.3 節 を参考にして，プログラム部分を`\begin{verbatim}` と `\end{verbatim}` で囲めばプログラムがそのままの形で打ち出される．

(0) V.8.2 節を参考にして，V.6.3 節 の `lln2.c` を書き換えて，結果を画面でなくファイル `lln3.dat` に書き込むプログラム `lln3.c` を作り（宿題２），実行し，プログラムと結果を報告せよ．`seed` は，学籍番号の下 2 桁を `nm` とするとき `5nm001` とせよ．（例：78 番の人は seed= 578001.）

(1) V.6.4 節 の `clt.c` を書き換えて，`lln2.c` のように途中結果を全て書き出さず数カ所だけ出力し，大きな $n$ まで計算するようにし，かつ，V.8.2 節を参考にして，結果をファイル `clt3.dat` に書き込むようにしたプログラム `clt3.c` を作り（宿題２），実行し，プログラムと結果を報告せよ．`seed` は，学籍番号の下 2 桁を `nm` とするとき `5nm001` とせよ．（例：78 番の人は seed= 578001.）

[3] 次のうち一つ以上に答えよ．

(a) V.6.3 節 で，$n$ 個の乱数の平均（$n$ 歩のランダムウォーク $S_n$ に対して $S_n/n$）が $b \times n^{-a}$ に近い速さで 0 に収束するとき，減少の指数 $a$ のおおよその値を `lln2.c` の結果から読みとりやすいという趣旨の説明が書いてある（宿題２）．実際に `lln2.c` を実行し，そこから $a$ のおおよその値を読みとる方法を説明し，かつ読みとった $a$ の値を報告せよ．

(b) V.6.3 節 の大数の法則から `lln.c` や `lln2.c` の乱数列の平均 $S_n/n$ が,（正しい乱数ならば） $n$ が大きくなると 0 に近づくことが期待されるが，どのような速さでゼロに近づくか．V.6.4 節 の最後の部分の議論に基づいて $n$ の何乗程度になるか，その指数を報告し（宿題２），`lln*.c` の実行による実測で検証せよ．

(c) 重複対数の法則によれば，V.6.5 節 の `lil.c` の結果は 1 に近づくべき量を計算しているはずであるが，例示してある結果を見ると，1 より減少する傾向があるように見える．その理由を報告せよ（宿題２）．また，この結果から重複対数の法則（右辺が 1 になること）を確認するためには数値計算結果をどう分析すればよいか？（宿題２）実際に数値計算を実行し，その結果に基づいて分析を行え．

(d) V.6.5 節 の重複対数の法則をランダムウォークの場合に証明するか，証明の載っている文献を探せ．単行本の場合は，著者，題名，発行年，第何版か，発行会社，章節番号，雑誌論文の場合は，著者，題名，発行年，雑誌名，第何巻か，証明のページ範囲，Web page の場合は証明の掲載されている URL (`http://...`)，をそれぞれ報告すること（宿題２）．また，どのようにして探し当てたかも報告せよ．

(e) V.7.1 節 の (V.7.7) を証明するか，証明の載っている文献を探せ．単行本の場合は，著者，題名，発行年，第何版か，発行会社，章節番号，雑誌論文の場合は，著者，題名，発行年，雑誌名，第何巻か，証明のページ範囲，Web page の場合は証明の掲載されている URL (`http://...`)，をそれぞれ報告すること（宿題２）．また，どのようにして探し当てたかも報告せよ．