確率変数の変換

- 一様乱数 X = U[0, I) (0 ≤ X < I) から、様々な確率分布に従う乱数を生成できる
- 指数分布に従う乱数、正規分布に従う乱数、などなど
- 具体例：一様乱数 X = U[0, I) に対して、
  - Y = X + I は U[I, 2) に従う（自明）
  - Y = 2X は U[0, 2) に従う（自明）
  - Y = X^2 はどんな分布になるか？
  - 任意の変数変換 Y = f(X) により、Y はどのような分布に従うのか？

1

今回は、一様乱数 U[0, 1) を変数変換することにより、別の分布に従う乱数を生成する事に取り組む。新たな確率変数 Y = 2 U[0, 1) は、0から2の一様乱数であることは自明（ではどうやってそうであることを示すか？）。また、Y = U[0, 1)^2 はどのような分布に従うのだろうか？

確率変数の変換

- 一様乱数 X = U[0, I) を変換 Y = f(X) により変換する
- Y の分布はどのようなものになるか？（確率変数の変数変換）
- X の分布 p(x) と Y の分布 q(y) の関係

$$\mathrm{Prob}[x < X < x + dx] = p(x)dx \qquad p(x) = 1 \quad 0 \le x < 1$$

$$\mathrm{Prob}[y < Y < y + dy] = q(y)dy$$

$$p(x)dx = q(y)dy$$

$$q(y) = p(x)\left|\frac{dx}{dy}\right|$$

2

一様乱数 X = U[0, 1) を変数変換 Y = f(X) により X —> Y に変換することを考える。Xの分布（確率密度関数）を p(x)、Yの分布（確率密度関数）を q(y) とすると、x < X < x + dx である確率と y < Y < y + dy である確率は等しいので、p(x) dx = q(y)dy を得る。ここで X は U[0, 1) であれば、p(x) = 1 for 0 ≤ x < 1, p(x) = 0 for otherwise である。この関係を用いると Y の分布は最終的に次式で与えられる。つまり、y = f(x) の逆関数 x = f^{-1}(x) の微分を用いて表すことが出来る。

指数分布する乱数

- 一様乱数 X = U(0, 1] を Y = – (log X)/γ で変換して得られる乱数Y はパラメータ γ の指数分布に従う。log 0 = – ∞ なので 0 は除く

$$Y = -\frac{\log U(0,1]}{\gamma} = -\frac{\log(1 - U[0,1))}{\gamma}$$

$$\mathrm{Prob}[y < Y < y + dy] = q(y)dy$$

$$q(y) = \gamma e^{-\gamma y} \quad y \geq 0$$

$$E[Y] = \int_0^\infty yq(y)dy = \frac{1}{\gamma}$$

3

このことを用いると、Y = –(log U(0, 1])/g (g > 0) は、パラメータ g の指数分布に従うことがわかる（自分で確認せよ）。パラメータ g の指数乱数の平均値は g の逆数となる。

指数分布する乱数の生成

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

# define GAMMA 2.0    <— 指数分布のパラメータ γ を設定

extern void init_genrand(long);
extern double genrand_real2(void);

int main(void)
{
  long seed;
  int i;
  double rand, rand2;

  seed = (long)time(NULL); // seed の設定


  init_genrand(seed); // seed で初期化

  for(i=0; i<3000; i++){
    rand = 1 - genrand_real2(); <— 1 – U[0, 1) = U(0, 1]
    rand2 = -log(rand)/GAMMA;
    printf("%.20f\n", rand2);
  }

  return 0;
}
```


4

メルセンヌ・ツイスタを用いたパラメタ GAMMA に従う指数擬似乱数はこのようにして生成可能。log 0.0 = -無限大、を踏まないように、U(1, 0] = 1 – U[0, 1) を用いている。自分でプログラムを実行して確認すること。

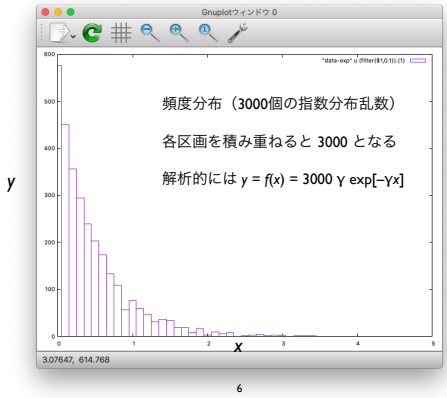
指数分布する乱数の生成

```
% cc sample-MT-exp.c ran-MT.c        コンパイル・リンク

% ./a.out > data-exp          実行結果をファイルに書き出す

% gnuplot             gnuplot で頻度分布を表示

   G N U P L O T
   Version 5.2 patchlevel 8    last modified 2019-12-01

   Copyright (C) 1986-1993, 1998, 2004, 2007-2019
   Thomas Williams, Colin Kelley and many others

   gnuplot home:     http://www.gnuplot.info
   faq, bugs, etc:   type "help FAQ"
   immediate help:   type "help"  (plot window: hit 'h')

Terminal type is now 'qt'
gnuplot> filter(x,y)=int(x/y)*y     <— 頻度分布表示のためのフィルター設定
gnuplot> plot "data-exp" u (filter($1,0.1)):(1) smooth frequency with boxes
gnuplot>
                                        <— 頻度分布を刻み幅 0.1 で表示
```


5

メルセンヌ・ツイスタ本体と一緒にソースプログラムをコンパイル、実行。
実行結果をファイルに書き出し、gnuplot を用いて頻度分布を描く。このとき、このような filter を定義すると簡単に頻度分布を描くことができる。

指数分布する乱数の生成



y

頻度分布（3000個の指数分布乱数）

各区画を積み重ねると 3000 となる

解析的には $y = f(x) = 3000\,\gamma\,\exp[-\gamma x]$

6

ここでは3000個の指数乱数の頻度分布を描いている。頻度なので、各区画に落ちた数を合計すると3000になる。パラメタgの指数分布（確率密度関数）は積分すると1になるので、これに乱数の数を掛けた 300 g exp[-g x] のグラフと（ほぼ）一致するはず。より多数の指数乱数を生成すれば、実際の頻度分布と解析解とはより一致する（自分で確認すること）。

モンテカルロ法

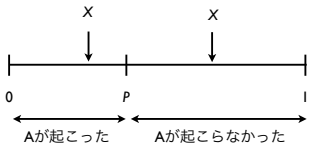
- 乱数を用いてシミュレーションや数値計算を行う手法の総称
- 物理学や生物学などのシミュレーションに良く用いられる
- 具体例：コイン投げ、ランダムウォーク（乱歩）、など、確率論的な事象の変化をアルゴリズムとして記述して実行する

7

ここまでは、一様乱数 U[0, 1)を生成する事で、任意の区画の一様乱数 U[a, b)、ならびにパラメータ g > 0 の指数乱数を生成した。次に、乱数を用いた様々なシミュレーションについて学ぶ。
乱数を用いてシミュレーションや数値計算を行う手法を総称してモンテカルロ法と呼ぶ。
物理学や生物学などのシミュレーション事件によく用いられる。
身近な例としては、コイン投げ（コインの裏表のどちらがでるか？）、ランダムウォークなどがある。

確率的な事象のプログラム実装

- 確率 *P* で起こる事象 A をプログラムとして実装（$0 \le P \le 1$）
  - [0, 1) の疑似一様乱数 *X* を生成
  - *X* < *P* なら、事象 A が起こったと見なす。そうでなければ起こらなかったと見なす（ベルヌイ試行）



X は区間 [0, 1) で一様

8

まず最初に、ある確率 P で起こる事象（起こるか、起こらないか、のどちらか）を確率論的に実装する方法を説明する。コイン投げの場合 P = 0.5。一様乱数 X = U[0, 1) を生成し、X < P なら事象が起こった、そうで無ければ起こらなかった、と見なすことで確率 P で起こる事象を実装できる（ベルヌイ Bernoulli 試行の実装）。Cで書けば以下のようになる。

```
int event;
double rv;
rv = genrand_real2();
if( rv  < P )
  event = 1;
else
  event = 0;
```

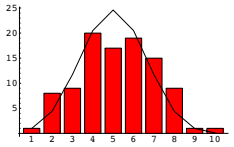
二項分布する乱数生成（コイン投げ）

- 正しく造られたコインは裏表がでる確率は $P = 1/2$ である。
- n 回コインを投げたとき、i 回表がでる確率（$i = 0, 1, 2, \ldots, n$）は、二項分布で与えられる（ベルヌイ試行の複数回繰り返し）

$$P_n(i) = {}_n\mathrm{C}_i \left(\frac{1}{2}\right)^n$$

具体例：$P = 1/2$, $n = 10$ を 100 回繰り返したとき、表が出た回数

{7, 7, 9, 8, 4, 7, 8, 3, 7, 3, 1, 4, 2, 4, 5, 5, 6,
3, 8, 7, 5, 2, 6, 4, 4, 6, 3, 3, 5, 6, 2, 5, 8, 2,
6, 2, 6, 4, 6, 7, 10, 4, 5, 8, 5, 4, 4, 5, 4, 5, 3,
3, 8, 5, 5, 6, 5, 5, 4, 6, 6, 4, 2, 7, 2, 6, 6, 6,
4, 7, 5, 6, 3, 6, 7, 3, 6, 6, 8, 5, 6, 4, 7, 4, 2,
4, 5, 8, 7, 4, 4, 5, 7, 4, 4, 7, 7, 8, 6, 7}

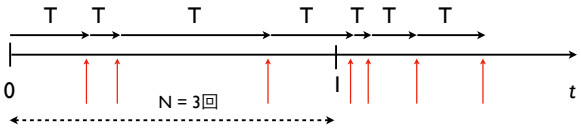


9

ベルヌイ試行を複数回繰り返すと二項分布に従う乱数を生成することが可能となる。

ポアソン分布する乱数

- 発生率（単位時間の発生回数）が γ である出来事が起こるまでの待ち時間 T はパラメータ γ の指数分布に従う（Poisson過程）



単位時間内に起こる回数 N はパラメータ γ のポアソン過程に従う

$$P_\gamma(N = n) = \frac{e^{-\gamma}\gamma^n}{n!} \quad n = 0, 1, 2, \cdots$$

10

単位時間の平均発生回数が g > 0 である出来事が起こるまでの待ち時間 T はパラメータ g の指数分布に従う（ポアソン過程）。このことを用いると、ポアソン分布に従う乱数（非負の整数値）を生成することが出来る。T = – (log U(0, 1])/g を生成し、時刻 t = 0 を T だけ進め (t += T)、初めて t が 1 を超えるまでに起こった出来事の回数がパラメータ g のポアソン分布に従う乱数となる。この図の場合 N = 3 となる。

問題 3

- 0 から 1 の一様乱数 X = U[0, 1) を下記の変数変換を施して得られる確率変数Y が満たす確率密度関数 q(y) を求めよ。疑似一様乱数を用いて、この結果を確かめよ
  - Y = – log X
  - Y = $\sqrt{X}$
- 2つの一様乱数 X = U[0, 1), Y = U[0, 1) を用いて以下の変換で得られる確率変数 Z は、平均が0、分散が1の正規分布に従う（ボックス - ミュラー法）。Z を多数を生成し確率分布を描いて確認せよ

$$Z = \sqrt{-2\log X}\cos 2\pi Y$$

11

X = U(0, 1] に対して、変数変換 Y = –log X（logは自然対数）、Y = Sqrt(X)（平方根）、によって生成される乱数の分布を求めよ。実際にメルセンヌ・ツイスタを用いて生成した乱数の頻度分布と比較すること。
一様乱数 U[0, 1) から正規分布（ガウス分布）する乱数を生成する方法としてボックス・ミュラー法がある。この用法を用いて擬似乱数 Z を多数生成して頻度分布を描け。

問題 4

- コインを *n* 回投げる試行を *k* 回繰り返す。各試行で表が出た回数を *i* とする。*i* をファイルに書き出せ。
- 試行数 *k* を十分大きくとったとき、*i* の分布図を描け。また理論分布と比較せよ。*n* の値は適当でよい。

1回の試行で表が出た回数を戻り値として返す関数を定義

```
int rv_binomial(int num, double prob)
{

  .....
  return count;
}
```


12

二項分布に従う擬似乱数を計算する関数 rv_binomial(int num, double prob) を定義して多数の擬似乱数を生成せよ。引数 num は繰返し回数、prob はベルヌイ試行でイベントが起こる確率（コインの表が出る確率）とする。

問題 5

- パラメータ λ > 0 で指数分布する乱数 T を生成する事により、パラメータ λ に従うポアソン分布する乱数 N を多数（数千個）生成し、理論と比較せよ

下記の関数を定義して用いよ

```
int rv_poisson(double lambda)
{

  .....
  return count;
}
```


同じく、指数乱数を生成する事で、パラメータ lambda のポアソン分布に従う乱数（非負の整数）を生成する関数 rv_poisson(double lambda) を定義して、多数のポアソン分布に従う乱数を生成し、分布を調べよ。lambda > 0 の値は適当で良い。