文献E2)

乱数発生プログラム

井田昌之

1. 乱数とは？

Tiny BASIC をはじめとして，マイクロコンピュータ上の BASIC言語には“乱数”を発生させるルーチンが組み込まれているものが多い．これらのルーチンがなければ，BASIC で書かれた各種のゲームの魅力を半減させることになるだろう．逆にいえば RND などの名をもつ乱数発生ルーチンは，ゲームのプログラムに意外性を与える働きをしている．たとえば，あまりの楽しさで筆者をタイプライタの前に4時間も釘付けにした宇宙戦争のプログラム（245 ページ参照）も，やっつけるべきクリンゴンの数やその位置，あるいは障害となる星の数や位置，宇宙嵐がいつくるかなどすべてが定まっていたのなら，何度も繰り返して挑戦する気にはならなかっただろう．

乱数発生ルーチンはこのように意外性をもった状況の設定によく用いられており，“スーパーマーケットの混雑状況を模擬し解析する”といったシミュレーション・プログラムなどには不可欠のものとなっている．また計算機を離れてみても，工場の抜き取り検査のサンプリング間隔を決めるためなどに乱数表というものがしばしば用いられている．この数表は乱数発生ルーチンにより簡単に作成することができる．

乱数の用途の詳細な説明は別の機会に譲ることにして，最大の関心事である“よい乱数とは何か？”について，もう少しふれてみることにしたい．

まず第一に，“5という数は乱数か？”という設問がいま仮にあるとしよう．こういう問い自身が妥当なものであるだろうか？

乱数の意味を文字通り“乱れた数”と考えたとき，それは乱数の真の意味からそれほど遠くはないので，上の設問を次のように置き換えてみることができる．

“5という数は乱れた数か？”

もし1，2，3，4と順に数が連なっており，次に5があげられたときならば，この設問の答えは感覚的にいって“No”であろう．しかし，6，2，1，3の次に5があげられたときならば，“Yes”と感じることもできよう．いずれにしても，単独の数に対して“乱れているか？”と聞く設問はそれ自体無意味であるが，付帯状況によってはその気持を推察することも不可能ではない．

それでは，その付帯状況とは何であろうか．

ある数に乱数らしさを感じるのはどういう場合であろうか．それは，基本的には繰返しのなかの意外性である．正しくいえば，数列の中の無規則性と一様性（等出現性）である．この2つの性質（詳細は後述する）をもった数列を乱数列と呼び，各要素を乱数という．たとえば，サイコロを振って出た数は無規則な数列を構成し，かつその値は1から6までの数値が等しい確率をもって出現するということがいえそうなので，乱数とみなすことができる．

2. コンピュータ上に“でたらめな数”を発生できるか？

コンピュータはあらかじめ作成したプログラムに従って忠実に動作することを特徴とするので，誰にも（プログラム作成者にも）まったく予期できないようなでたらめな数列を手続き的に発生することは不可能である．もっとも，物理的な確率現象などを計測しその数値を AD

変換して，計算機に送り込むような周辺機器を作成し接続すれば，まったく予期できない数よりなる数列を構成することは可能である．しかし，そうした数列には再現性がないので，不便さと乱数としての性質のチェックなどに問題があり，あまり実用的とはいえない．

このため，計算機上で扱う乱数の“でたらめさ”は限定されたものであって，使用目的に応じて乱数列のでたらめさを確認（検定）することが必要である．こうした乱数を擬似乱数という．擬似乱数は，その発生ルーチンの内部を知らないものにとっては，自分のソース・プログラムを見ているだけでは，次にどんな数が飛び出すか予測できないが，発生手続きを知っているものにとってはその数列は既知のものとなる．

3. “でたらめさ”とは？

計算機プログラムにより発生させる乱数列のもつべき性質は，経験的に次のような点にまとめられることが一般に知られている．

① 定められた区間内に一様にばらついている．

等出現性ないしは等頻度性ともいわれる．正しいサイコロは1から6までの目が等しい確率で出現するのと同様に，たとえば0から9までの値を取りうる整数乱数は，1000回の乱数発生においてはその中に0が100個，1が100個，……，9が100個ずつ含まれているのが望ましい．

② 統計的によい性質をもつ．

等出現性だけを問題にすると，たとえば，0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 1, …, 9, 0, …，といった数列から100個とか1000個とか（一般に10*n*個）を取り出すと，その中の0から9までの各数の出現頻度は*n*個ずつで等出現性を満たしてしまう．そのために系列相関とか，数の現われ方の規則性などがないと考えないと実用的な乱数とはいえなくなる．

③ 再現性がある．

乱数列の統計的性質のテストを，任意の時点で行ないうることが望ましい．また，シミュレーションなどにおいては途中経過の再現など，まったく同じ乱数列を繰り返して用いたい場合がある．

④ 周期がなるべく大きい．

擬似乱数は，その発生原理から同じ数列を完全に繰り返す．この周期は，試行（乱数の使用）回数に比して充分大きなものであることが望ましい．小さな周期をもつ乱数は，多くの場合によい結果をもたらさない．

⑤ 高速である．

モンテカルロ法，シミュレーションなどに乱数を利用する場合，その発生個数は解に影響を与えるので，かなり多くの乱数を必要とする．このため，乱数発生ルーチンの速度が遅いと，膨大な計算時間を全体として必要としてしまうこともありうる．

⑥ 所要メモリ数が少ない．

乱数発生ルーチンがよい性能をもったものであっても，メモリ上に乗らないほど大きなものであったなら，実用価値はなくなってしまう．特にマイクロコンピュータの場合には，速度を損わない程度にプログラム・サイズを縮めることが必要である．

これらの性質は多分に経験的であり，乱数列の使用においては，目的に応じて重点をおく性質を考えなおす必要がある．

4. 乱数発生ルーチンの実例

Palo Alto 版 Tiny BASIC の乱数発生ルーチンは，BASIC インタプリタから RND(X) という形式の関数コールに対応して呼び出される．

$1 \leq X \leq 32767$

$1 \leq RND(X) \leq X$

そのプログラムの中核部は，次のようになってい

```
             ⋮
        LHLD  RANPNT
        LXI   D,LASTM
        CALL  COMP
        JC    RA1
        LXI   H,0;(H,L)>(D,E)
RA1:    MOV   E,M
        INX   H
        MOV   D,M
        SHLD  RANPNT
             ⋮   (以下省略)
```

変数 RANPNT は，システム・プログラムが書き込まれた領域（0番地から LASTM 番地まで）を示すポインタで，そこから2バイトがそのときの乱数として D, E レジスタへ順にとられていく．つまりメモリ中の命令を乱数表のかわりに利用している（図1）．そして，D-E レジスタに与えられる乱数 x_i をこの後指定された範囲に剰余を用いておとし，RND(X) の値とする．

こうした方法では，その手軽さの反面，プログラム中の命令コードはかたよりがあり，3章で示した性質のうちの①等出現性，②統計的な良い性質，④大周期性などが満たされず，ゲーム用の乱数程度以上の応用には危険がある．

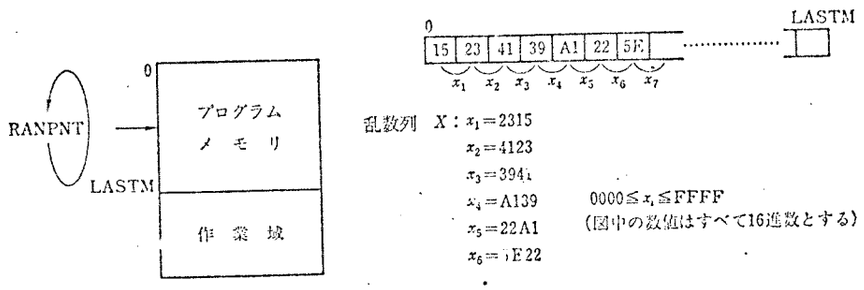


図1　プログラム自身を乱数表とみなした乱数発生方法.

5.　乱数の発生方法

現在広く用いられている乱数発生方法は，線形漸化式を用いて順次乱数列を発生させる方法である．これを一般に合同法という．合同法のなかで一番簡単な方法は，フィボナッチ法と呼ばれるもので，直前の二項の和により次の項を決定するものである．その一般形は次のとおりである．

$$x_{n+1}\equiv x_n+x_{n-1} \pmod p$$

ここで mod p は，p で割った余りを取るという意味である．たとえば $x_1=2$, $x_2=5$, $p=13$ とすれば，発生数列は

2, 5, 7, 12, 6, 5, 11, 3, 1, 4, 5, 9, … となってゆく（$x_5=7+12 \pmod{13}=19 \pmod{13}=6$）．フィボナッチ法より一般によい性質をもつとされる方法は，乗積合同法と混合合同法である．乗積合同法は

$$x_{n+1}\equiv\lambda x_n \pmod p$$

により，混合合同法は

$$x_{n+1}\equiv\lambda x_n+\mu \pmod p$$

により乱数列を発生させる．

たとえば，$\lambda=5$, $p=8$, $x_0=1$ による乱数列は，乗積合同法では

5, 1, 5, 1, 5, …

となり，周期は2である．

ところが，混合合同法（$\mu=1$ とする）では

6, 7, 4, 5, 2, 3, 0, 1, 6, 7, 4, 5, …

となり，周期は8となる．$p=8$ であるので，この方法は周期が最大限に長く，乗積合同法の場合と比較してかなり良い発生方法であることが感じられる．発生した乱数の統計的性質は，混合合同法が乗積合同法より必ずしも優れているとはいえないことがすでに指摘されている．しかし混合合同法のほうが同じpに対して長い周期を発生することができるので，マイクロコンピュータの場合には，より望ましいと思われる．両者の周期は解析されており，乗積合同法の場合には，最長周期 2^{d-2} が次の場合に与えられる．

$$x_{n+1}\equiv\lambda x_n \pmod{2^d} \qquad d\geqq 3,\ x_0 \text{ は奇数}$$
$$\lambda\equiv 3 \text{ または } 5 \pmod 8$$

混合合同法の場合には，最長周期 2^d が次の場合に与えられる．

$$x_{n+1}\equiv\lambda x_n+\mu \pmod{2^d}$$
$$\lambda\equiv 1 \pmod 4,\ \mu\equiv 1 \pmod 2,\ x_0 \text{ は任意数}$$

6.　でたらめさの検証

6.1　統計的仮説検定

サイコロを投げる場合を考えてみよう．あるサイコロを2回続けて投げたところ，全部1の目がでたとする．そのとき，サイコロを投げた人は“偶然だ”と思うに違いない．しかし，さらに続けて2回投げ，合計4回投げた結果が全部1であったとき，サイコロが正常であると，なおも信じる人は少ないだろう．正しいサイコロを4回続けて投げて全部1が出る確率は

$$\left(\frac{1}{6}\right)^4=0.0008$$

となり，きわめてまれなことである．したがって，4回続けて投げて全部1がでたサイコロが正常でないと判断することは，多くの場合は正しいと考えられる．

このように，ある仮説（この場合“サイコロは正常である”）のもとで，非常に確率の小さい結果がただ1回の実験で起こったとすれば，その仮説を認めないと判断する．多くの場合，仮説が正しくないと判断する信頼限界は 0.05 とか 0.01 という確率がとられる．このときの判断には，大きいときには5％ないし1％の判断の誤り（正しいのに正しくないと判断する誤り）があることに

表1　サイコロの目の数の和

和	④	⑤	⑥	7	8	9	10	11	12	13	14	15
確率	1/1296	4/1296	10/1296	20/1296	35/1296	56/1296	80/1296	104/1296	125/1296	140/1296	146/1296	140/1296

和	16	17	18	19	20	21	㉒	㉓	㉔
確率	125/1296	104/1296	80/1296	56/1296	35/1296	20/1296	10/1296	4/1296	1/1296

表2　乱数の一例

区間	0 ～ 0.09	0.10 ～ 0.19	0.20 ～ 0.29	0.30 ～ 0.39	0.40 ～ 0.49	0.50 ～ 0.59	0.60 ～ 0.69	0.70 ～ 0.79	0.80 ～ 0.89	0.90 ～ 0.99	合計
度数	31	25	22	17	24	18	27	31	28	27	250

なる．（このような判断の誤りを第1種の誤りという．これに対して，仮説が真でないのに受け入れてしまう誤りを第2種の誤りという．）

このような手続きにより仮説を棄てたり，受け入れたりする方法を，統計的仮説検定という．そして上に示した判断の誤り率を危険率ないしは，仮説をすてる基準という意味で有意水準という．また棄却される結果の全体を棄却域と呼ぶ．また，仮説検定に用いられる仮説を帰無仮説ともいう．

確率分布がわかっている統計量を与えて，その実現値が棄却域の中にはいっていれば，“仮説を捨てる”と判断し，その中になければ“仮説を捨てない”と判断する．棄却域は，両側または片側にとられる．

例として，サイコロが正常であるという仮説をサイコロを4回投げることにより，危険率5％で検定することを考えてみよう．

サイコロを4回投げたときの目の数の和は，4から24までの値をとりうる．その確率分布は表1のようになる．

危険率を5％と設定することは，全体の確率を1としたときに 0.05 の確率で起こる事象がただ1回の試行により起こることは統計的にみて，まずありえないと考えることと等しい．このときの棄却域は

$$0.05>\frac{1}{1296}+\frac{4}{1296}+\frac{10}{1296}+\frac{10}{1296}+\frac{4}{1296}+\frac{1}{1296}=0.023$$

より，表1の○印のついたところになる．すなわち，目の和が 4, 5, 6, 22, 23, 24 の場合である．サイコロを4回投げてその目の和が 4, 5, 6, 22, 23, 24 であれば，危険率5％で，仮説を棄て，そのサイコロを正常とはいえないと判断する．

6.2　カイ二乗検定

統計的検定には等平均・等分散の検定（二つの統計量の間に差異はあるか？　などのテスト），独立性の検定（二つの事象のあいだには関連があるか？　など）および適合度の検定（二つの分布のあてはまりの良さをみる）などがある．乱数の検定は，このうち適合度検定という部分にはいり，実際にはカイ二乗統計量を計算し，それらが理論分布であるカイ二乗分布とどれだけ適合しているかを調べる．この方法はカイ二乗検定と呼ばれ，広く用いられているものである．

カイ二乗検定を行なうには，次の手順に従えばよい．

① 検定したい乱数を適当な数だけ発生させる．多くの場合は 1000 個程度を発生させる．

② 乱数のとりうる区間を 10 とか 20 とかの小区間に区切って，乱数が各区間にはいる個数(度数)を調べて表にする．いま仮に，区間 (0, 1) の乱数 250 個を 10 個の区間に分けて表2のような結果が得られたとする．理論的には，各区間には 25 個ずつはいっているはずである．

③ 棄却領域の設定．250 個の乱数が各区間内に一様にちらばっている（定められた区間内に等確率で現われる）という仮説を検定するためには，カイ二乗分布を利用する．

$$\chi^2=\sum_{i=1}^{m}\frac{(n_i-np_i)^2}{np_i}$$

m：小区間の個数
n：乱数の総数
n_i：第 i 区間にはいっている乱数の数
p_i：第 i 区間にはいる確率

上式で導かれるカイ二乗統計量は，自由度 $(m-1)$ のカイ二乗分布に従うことが知られている．そこで，危険率＝0.05 としたときの棄却領域を設定する．この区間

表 3　典型的な棄却域

区間数	自由度 \ 危険率 α	0.05	0.01
8	7	(14.1, ∞)	(18.5, ∞)
10	9	(16.9, ∞)	(21.7, ∞)
16	15	(25.0, ∞)	(30.6, ∞)
20	19	(30.1, ∞)	(36.2, ∞)
32	31	(43.8, ∞)	(50.9, ∞)

は，数表により求められる．よく用いられている区間数は 8, 10, 16, 20, 32 などであるので，その場合の棄却域を表 3 に示す．

理解を増すためにカイ二乗分布の概形と片側検定の棄却域を図 2 に示す．

②で用いた例の場合，区間数 $m=10$ であるので，危険率を 0.05 とすれば，棄却域は (16.9, ∞) となる．

④ 実現度数と理論度数から，各小区間ごとに

$$\frac{(\text{実現度数}-\text{理論度数})^2}{\text{理論度数}}$$

を計算し，これを各区間にわたって加える．この値を χ^{2*} と呼ぶ．

$$\chi^{2*}=\sum_{i=1}^{m}\frac{(n_i-np_i)^2}{np_i}$$

ここで，n_i は第 i 区間にはいったものの個数，p_i は第 i 区間に落ちる確率とする．

例の場合には

$$\chi^{2*}=\sum_{i=1}^{10}\frac{(n_i-np_i)^2}{np_i}$$
$$=\sum_{i=1}^{10}\frac{(n_i-250\times 0.1)^2}{250\times 0.1}$$
$$=\frac{(31-25)^2}{25}+\frac{(25-25)^2}{25}+\frac{(22-25)^2}{25}$$
$$+\frac{(17-25)^2}{25}+\frac{(24-25)^2}{25}+\frac{(18-25)^2}{25}$$
$$+\frac{(27-25)^2}{25}+\frac{(31-25)^2}{25}+\frac{(28-25)^2}{25}$$
$$+\frac{(27-25)^2}{25}$$
$$=8.48$$

より $\chi^{2*}=8.48$ となる．

この値はカイ二乗統計量 $\chi^2=\sum_{i=1}^{10}\frac{(n_i-25)^2}{25}$ の一実現値とみなせるかを次にテストする．

⑤ 棄却するかどうかを判断する．実現値 χ^{2*} がもし棄却域の中にはいっていれば，仮説を捨てる．そうでなければ仮説を受け入れる．

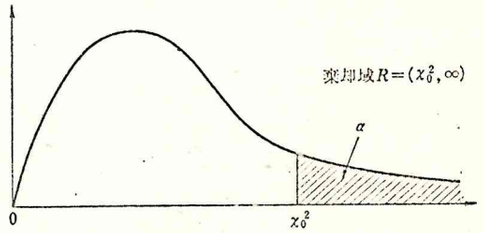


図 2　カイ二乗分布の概形

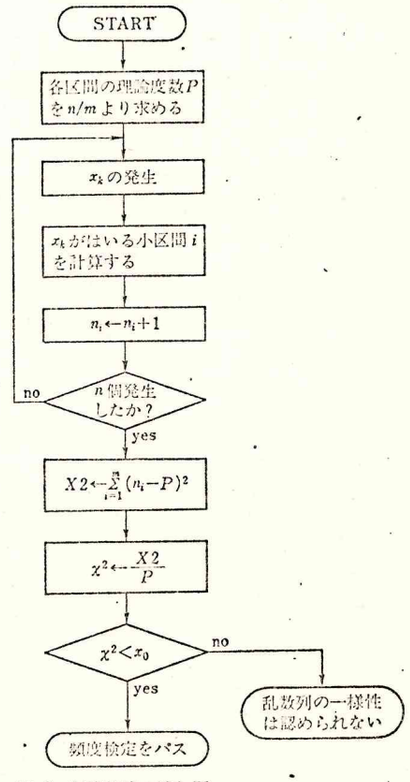


図 3　頻度検定の流れ図

例の場合は，$\chi^{2*}=8.48$，棄却域 $R=(16.9,\infty)$ であり，棄却域にはいっていない．したがって，仮説は受け入れられる．

この検定法は np_i をすべて等しいとしており，乱数列の等出現性を検定するのに用いられ，頻度検定ないしは度数検定と呼ばれるものである．乱数列 $x_k(k=1,\cdots,n)$ の頻度検定の手順は次の流れ図にまとめることができる（図 3）．

6.3　無規則性の検定——統計的に良い性質か？

無規則性をいろいろな角度からとらえる種々の検定法

が存在する．このうちのいくつかを紹介する．

① 連次検定（シリアル・テスト）

0 から 9 までの値をとりうる乱数に対して 2 つ組にしてその一様性をみる．たとえば，1, 5, 5, 3, 6, 7, 0, 2, 1, 3, …の場合は 15, 55, 53, 36, 67, 70, 21, 13, … という数を作り，各数の出現頻度をカイ二乗検定する．

② 系列相関検定

連次検定を一般化して 1 つとびとか 2 つとびなどの数の関連を検定する．おくれ k の系列相関 ρ_k は次の式で定義される．

$$\rho_k=\frac{E[x_ix_{i+k}]-(E[x_i])^2}{E[x_i^2]-(E[x_i])^2}$$

記号 E は平均値を表わすものとする．N 個の乱数列に対して上記の式により ρ_k^* を算出する．区間 (0, 1) に一様分布する乱数は，平均 $E[x_i]=1/2$，分散 $V[x_i]=1/12$ であるから，理論的には

$$\rho_k=\frac{12}{N}\sum_{i=1}^{N}x_ix_{i+k}-3$$

となる．N が充分大きければ，ρ_k の値は平均値 0 で，分散 $13/N$ の正規分布をするとみなせるので，算出した ρ_k^* に対して検定を行なう．

③ ギャップ・テスト

数列内の任意の数 d が次に現われるまでの間隔をギャップという．ギャップの長さ k は，0, 1, 2, … をとりうる．このとき，長さ k のギャップを真に無規則な数列が作る確率は

$$P(k)=(0.9)^k(0.1)$$

で表わされる．この確率をもとに，理論値と実際値との間のカイ二乗検定をすることができる．

④ 連の検定（上昇・下降連のテスト）

一様乱数 x_i に対して単調増加（単調減少）する回数 r を用いて検定する．これを上昇の連（下降の連）のテストという．長さ r の連が起こる頻度 N_r は

$$N_r=2n\frac{r^2+3r+1}{(r+3)!}-2\cdot\frac{r^3+3r^2-r-4}{(r+3)!}$$

となることが知られている．n は乱数の総数であり，r は

$$x_{n-1}>x_n<x_{n+1}<\cdots\cdots<x_{n+r}>x_{n+r+1}$$

により与えられる（下降の連の場合は逆の不等号）．

⑤ 連の検定（符号テスト）

乱数がその平均値（または中央値）より大か小かによって連を定義する．大きいものを“＋”，小さいものを“－”という符号で対応させる．たとえば区間 (0, 1) の乱数を考えると

0.542, 0.238, 0.625, 0.783, 0.151, 0.011, …

という数列に対しては

＋－＋＋－－…

という符号連ができる．この場合，長さ 1 の符号連が 2 つ，長さ 2 の符号連が 2 つとなっている．連の総数 K は，－符号をもつ数の度数を N_-，＋符号をもつ数の度数を N_+ とすれば

$$\text{平均値}=\frac{2n_+n_-}{n}+1$$

$$\text{分散}=\frac{2n_+n_-(2n_+n_--n)}{n^2\cdot(n-1)}$$

なる正規分布に従うことが知られている．正規分布表は平均 0，分散 1 であるので，$(K-\text{平均値})/\sqrt{\text{分散}}$ という変形を行ない，これを z とする．この z の値を用いて検定を行なう．

7.　他の分布に従う乱数の発生

いままで乱数と呼んできたものは定められた区間内を一様に分布するもので，厳密には一様乱数と呼ばれるものである．この章では，この一様乱数をもとに任意分布の乱数を発生する方法を紹介する．

① 指数分布

偶然事象が起こる時間間隔などは指数分布をすることが知られている．指数乱数の発生は，逆変換法と呼ばれる方法がよく用いられる．平均 μ の指数乱数 e_i は，区間 (0, 1) の一様乱数 r_i を用いて

$$e_i=-\mu\cdot\log r_i$$

により直接計算される．

② 正規分布

正規分布は，非常によく用いられる確率分布である．中心極限定理の応用として，正規乱数を簡単に作成することができる．たとえば

$$z=r_1+r_2+\cdots+r_{12}-6$$

は，平均 0，分散 1 の標準正規分布に従う正規乱数をなす．この z から平均 e，標準偏差 s の正規乱数 x は，次のようにして計算できる．

$$x=z\times s+e$$

このやり方は分布型の端のほうは切り捨てられており，それが問題となる場合には他の方策をほどこす必要がある．

8.　各種の一様乱数の発生

8.1　プログラム動作環境について

これから示すプログラムは，ミニコン FORTRAN で書かれた検定ルーチン以外は，すべて筆者らの手元にあるマイクロコンピュータでテストしたものである．このマイクロコンピュータは，LISP マシン ALPS/I として

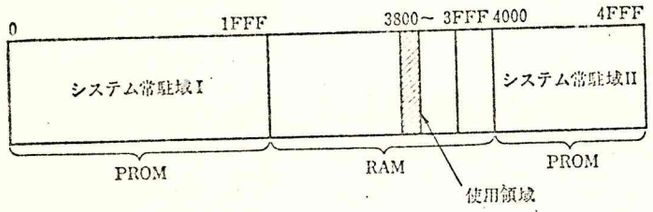


図 4 LISP マシンのメモリ構成

作成され，20K バイトの領域をもっている（図 4）．

しかし以下のプログラムは，このうちの 2000 番地(16進) から 256 バイト程度の領域しか使用していない．またこのなかから呼び出しているルーチンは，浮動小数点演算パッケージとサンプル・プログラム内で使用している 2 進16進変換ルーチン BINHEX, H-L レジスタの内容のヘキサプリント・ルーチン PUTHL および"RST2" 命令で呼び出される 1 字印字ルーチンだけである．

8.2 8 ビット乱数ルーチン RND 8（図 5：201ページ）

最初に示すルーチンは，8 ビットの 2 進数（0～255 の値）での乱数発生手続きである．発生方法は，次に示す式により混合合同法で行なわれる．

$$x_{n+1} \equiv 5 \cdot x_n + 1 \pmod{256}$$

RND 8 では x_n がレジスタ C.にはいっており，この数をもとに x_{n+1} を計算し，レジスタ C にいれなおす．この流れ図を図 6 に示す．

レジスタ C の内容（x_n）をレジスタ A に移し，RAL 命令により 2 倍する．2 回の RAL により 4 倍になるが，下 2 ビットは無意味なので 0FC と AND をとり，下 2 ビットをゼロにリセットする．次の ADD により，レジスタ A に $5 \cdot x_n$ ができる．このあと 1 を加え，レジスタ C に $5 \cdot x_n + 1 \pmod{256}$ を与える．

RND 8 を使用するサンプル・プログラムを図 7（201ページ）に示す．このルーチンは，初期値 $x_0=101$ として乱数列の最初の 256 個を印刷するものである．

16 区間で頻度検定を行なうと，50 個の乱数では $\chi^2=14.08$，100 個で $\chi^2=10.88$ となっており，自由度 15，危険率 5 % の棄却域は (25.0, ∞) で，いずれも棄却域にはいらず，良い結果となっている．

8.3 8 ビット以下の任意区間乱数ルーチン RND 8 X（図 8：201ページ）

上記の RND 8 を用いた任意区間（たとえば 0 から 99 まで）の乱数を発生するルーチンを図 8 に示す．

RND 8 X ルーチンは，RND 8 により作られた値 x を

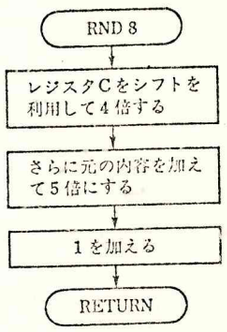


図 6 RND 8 の流れ図

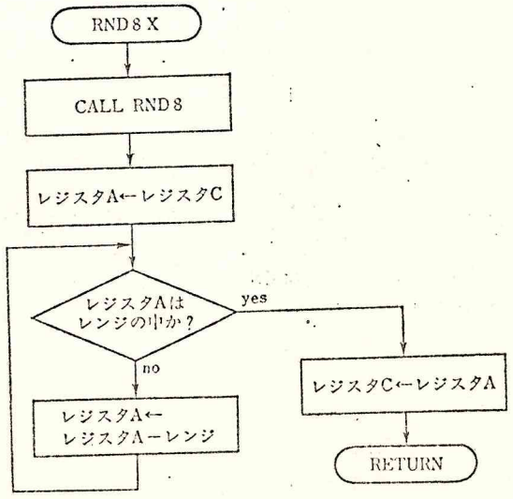


図 9 RND 8 X の流れ図

範囲 r で割った余りを乱数とする．たとえば

251, 228, 113, 50, 247, 208, 13, 62, 51, …

という乱数列が RND 8 により発生され，区間を 0～99 とすれば，RND 8 X の生成する乱数列は次のようになる．

51, 28, 13, 50, 47, 8, 13, 62, 51, …

RND 8 X はレジスタ B に区間の上限（0～99 までとしたいのなら 100）を入れて使用する．

流れ図を図 9 に示す．

RND 8 X を使用するサンプル・プログラムを図 10 に示す．これは図 8 とほとんど同じだが，レジスタ B をレンジを与えるために使用するので，ループのカウンタはレジスタ A を使用している．

8.4 16 ビットの乱数 RND 16（図 11：201ページ）

RND 8，RND 8 X は周期がいずれも短く，また 5 倍

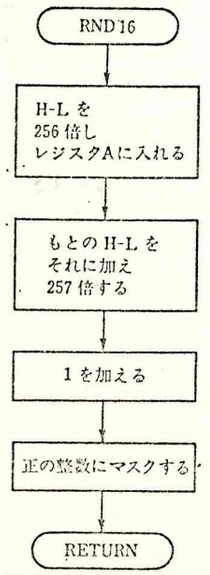


図 12 RND 16 の流れ図

して 1 を加えるという方式はそれほどよい統計的性質が与えられない．したがって，実用的な乱数としてはあまりすすめられない．本節で示す RND 16 は，16 ビット 2 の補数表現の演算を考え，このうち正の数（0 から 32767）をレンジとするルーチンである．このルーチンは，RND 8 と同じに 9 バイトしかかからないのに良好な統計的性質をもっている．RND 16 は次の式に従って乱数を発生させる．

$$x_{n+1} \equiv 257 \cdot x_n + 1 \pmod{2^{15}}$$

これは，任意の初期値に対して最高周期 2^{15} を与える．また定数倍するための 257 は，プログラムサイズを短くしかつ無規則性を増しうる値として選ばれている．

RND 16 によって発生される乱数列は，H-L レジスタにおかれる．この流れ図を図 12 に示す．

このルーチンは，3 ステップで 257 倍にするところに特徴がある．

ある数の 2 倍は，1 ビット左へシフト（桁移動）すればよい．（例：$(001011)_2$ の 2 倍は $(010110)_2$ である*.）一般に，2^n 倍したいときには n ビット左へシフトする．256 倍するためには，したがって 8 ビット左へシフトすればよいことになる．（例：$(0012)_{16}$ の 256 倍は $(1200)_{16}$ である．）

このことから，RND 16 の最初の 3 ステップ

MOV A, L

* $(\cdots)_2$, $(\cdots)_{16}$ は，（ ）の中が 2 進数，10進数であることを示す．

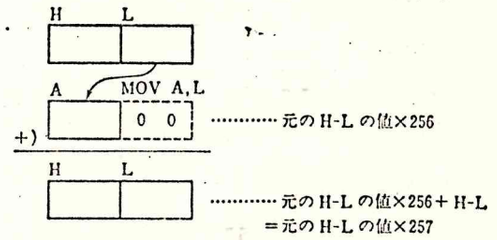


図 13 H-L の値を 257 倍する方法

ADD H
MOV H, A

は図 13 のような計算をしていることになる．

$x_0=1025$ としたとき，このルーチンは

1282, 1795, 2564, 3589, 4870, 6407, 8200, 10249, 12554, 15115, 17932, 21005, …

という数列を発生させる．この数列から n 個取り出したときの頻度検定は，$n=100$ ですでに $\chi^2=15.5$ で，危険率 5 % の棄却域 (25.0, ∞) にはいらず，n が大きくなるに従ってさらに良くなっており，検定には充分パスしている．

また，統計的性質をみるために連の検定（符号テスト）を行なってみる．そのプログラムを図 14 に示す．

図14のなかの主な変数の説明を次に示す．

MINUS: N_- のカウントのための変数
IPLUS: N_+ のカウントのための変数
J：0～32767 の乱数
K：連が続いているかどうかをチェックするためのスイッチ．一つ前の乱数が“－”であれば $k=1$ で，“＋”ならば $k=2$ である．
AK：連の総数
MAX, AMAX: 乱数の総数
Z：検定対象となる標準正規分布の一実現値

図14のプログラムの実行結果は

$Z=-0.124797$

が得られ，これは危険率 5 % の棄却域 $(-\infty, -1.96)$ および $(1.96, \infty)$ には属さない．したがって連検定をパスする．

図 15（202ページ）に RND16 のサンプル・プログラムを示す．このプログラムは，4096 個の乱数を発生させ，その平均値を求めて印刷する．

このプログラムは，スタック領域とレジスタ以外の特別な棄却域を使わずに値を求めるように組まれている．

また平均値はスタック・トップを H-L にポップアップし，H, L, D, E の順の 4 バイトの総和領域の値をシフ

bit

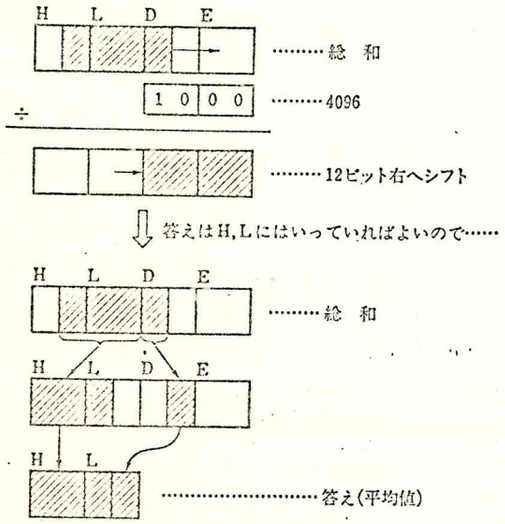


図 16 平均値を求める

```
10 IHALF=16384
20 MINUS=1
30 IPLUS=0
40 K=1
50 AK=0.0
60 MAX=1000
70 AMAX=MAX
80 J=1025
90 DO 10 I=1,MAX
100 J=J*257+1
110 IF(J.LT.0) J=J+32767+1
120 IF(J.GE.IHALF) GO TO 20
130 MINUS=MINUS+1
140 IF(K.EQ.1) GO TO 10
150 K=1
160 AK=AK+1.0
170 GO TO 10
180 20 IPLUS=IPLUS+1
190 IF(K.EQ.2) GO TO 10
200 K=2
210 AK=AK+1.0
220 10 CONTINUE
230 AN1N2=FLOAT(MINUS)*FLOAT(IPLUS)
240 AMIU=2.0*AN1N2/AMAX+1.0
250 SIGMA=SQRT(2.0*AN1N2*(2.0*AN1N2-AMAX)/(AMAX*AMAX*(AMAX-1.0)))
260 WRITE 100 AK,AMIU,SIGMA
270 WRITE 200 MINUS,IPLUS
280 200 FORMAT(" MINUS=",I6,"PLUS=",I6)
290 100 FORMAT(3F15.7)
300 Z=(AK-AMIU)/SIGMA
310 WRITE 500 Z
320 500 FORMAT("Z=",F15.7,"  VS.2.58(1PER) OR 1.96(5PER)")
330 STOP
340 END
```

図12と等価な乱数発生

図 14 連の検定プログラム

トにより除算を行なう（図16）．'DAD H' は H-L レジスタの左1ビットシフト，RRC はAレジスタの右1ビットシフトである．

このルーチンの実行結果は

4 0 0 1

がタイプライタ上に印字される．$(4001)_{16}=(16385)_{10}$ であり，0～32767 の乱数の平均の理論値は 16383.5 であるので，比較的良い値であろう．

8.5 RND 16 の応用 その1 RNDX（図 17：202 ページ）

RNDX は，レジスタ D-E で示すレンジをもつ乱数を H-L レジスタに発生させる．

RNDX の使用例を図 18（203ページ）に示す．図 18 のプログラムは，本質的には図17のものとまったく等しい．ただ，RNDX はレジスタ D-E に乱数の区間を指示しておかねばならないので，総和をとる領域はスタックの上位2つの4バイトの領域を用いている．

8.6 RND 16 の応用 その2 RND

一般に一様乱数といわれる乱数は，区間 [0, 1] の乱数である．このような乱数を作るためには，浮動小数点形式がなければならない．筆者らの利用している浮動小数点パッケージ（159ページ参照）を用いて作成した区間 [0, 1] の実数型一様乱数 RND を図 19（203ページ）に示す．

RND ルーチンの使用例として π の計算をさせてみる．図 20 に示すように一辺が1の正方形と半径1の四分単位円の面積比が1対 $\pi/4$ であることを利用する．

この流れ図を図 21 に示す．

この流れ図に従うプログラムを図 22（204ページ）に示す．この実行により，π は 3.10 まで求めることができる．（乱数列の中の 4001～5000 番までの 1000 個を使うと，3.14まで求められる．）

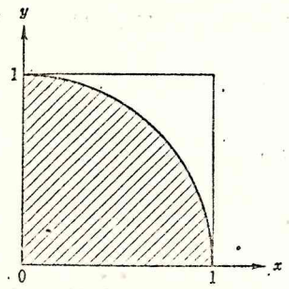


図 20 π の計算法

追 記

本稿を書き終えたのち，1977年11月号の BYTE 誌 218 ページに8ビットの乱数発生ルーチンのリストが掲載されているのを知った．

このルーチンは15バイト13ステップを要しており

$$x_{n+1}\equiv 13\cdot x_n+1 \pmod{256}$$

という式を用いており13倍は $4\cdot x_n$ を先に計算し，後にそれを2倍したものと x_n をさらに加えたものを加えて行なっている（$13\cdot x_n=(4\cdot x_n)\times 2+(4\cdot x_n)+x_n$）．

筆者の作成した RND 8 は5倍のみを行なっているので，初期値 x_0 の与え方によっては BYTE 誌の方法のほうがより良い統計的性質をもつかもしれない（しかし，筆者の RND 8 のほうがステップ数が短くて高速である）．

また，RND 8 はシフト命令の代わりに「ADD A」命令を用いることにより，もし必要ならばさらに高速かつコンパクトにすることができる．

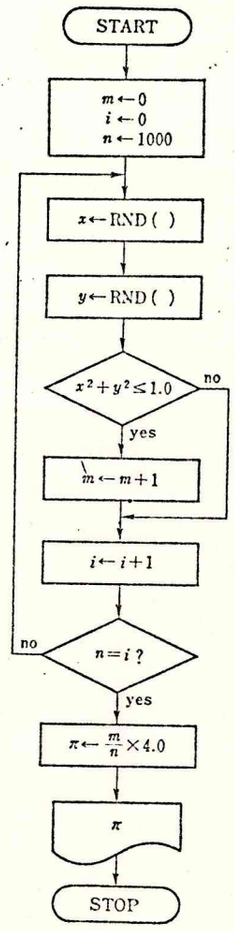
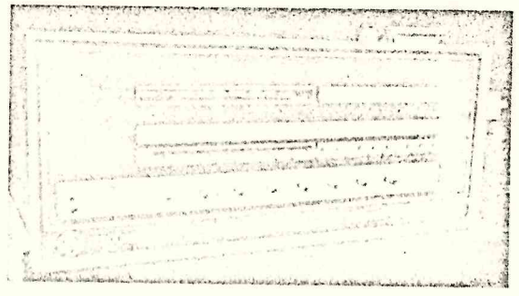


図 21 π の計算の流れ図

参考文献

1) 脇本和昌：乱数の知識，森北出版，1970.
2) 津田孝夫：モンテカルロ法とシミュレーション，培風館，1969.
3) ネイラー他，水野他訳：コンピュータシミュレーション，培風館，1971.



Altair 8800B（価格 421,000円）

```
                ;
                ;   *** RANDOM NUMBER GENERATOR  ***
                ;
3900                    ORG   3900H
                ;
                ;   ** RND8 ** 8BIT(0-255)
                ;
3900    79      RND8:   MOV   A,C
3901    17              RAL
3902    17              RAL
3903    E6FC            ANI   0FCH
3905    81              ADD   C
3906    3C              INR   A
3907    4F              MOV   C,A
3908    C9              RET
                ;
                ;   ** RND8 SAMPLE  **
                ;
                ;     PRINT 256 NUMBERS
3800    0600    R8TST:  MVI    B,0
3802    0E65            MVI   C,101
3804    CD0039  R8T1:   CALL  RND8
3807    CD2138          CALL  PUTR
380A    05              DCR   B
380B    C20438          JNZ   R8T1
380E    C9              RET
                ;
                ;   ** RND8X ** 8BIT
                ;
3909    CD0039  RND8X:    CALL RND8
390C    79              MOV   A,C
390D    90      R8X1:     SUB   B
390E    CA1539            JZ    R8X2
3911    D20D39            JNC   R8X1
3914    80                ADD   B
3915    4F      R8X2:     MOV   C,A
3916    C9                RET
                ;
                ;  ** RND8X SAMPLE **
                ;      PRINT 256 NUMBERS
                ;      RANGE=100
                ;
380F    0664    R8XTST: MVI   B,100
3811    0E65            MVI   C,101
3813    AF              XRA   A
3814    F5      R8XT1:  PUSH  PSW
3815    CD0939          CALL  RND8X
3818    CD2138          CALL  PUTR
381B    F1              POP   PSW
381C    3D              DCR   A
381D    C21438          JNZ   R8XT1
3820    C9              RET
                ;
                ;   ** RND16 ** 16BIT (0-32767)
                ;
3917    7D      RND16:  MOV   A,L
3918    84              ADD   H
3919    67              MOV   H,A
391A    23              INX   H
391B    7C              MOV   A,H
391C    E67F            ANI   7FH
391E    67              MOV   H,A
391F    C9            RET
```

図 5 RND8 ソース・リスト

レジスタ C の内容(x_n)をレジスタ A に移し，RAL 命令により2倍する．2回の RAL によって4倍になるが，下2ビットは無意味なので 0FC と AND をとり，下2ビットをゼロにリセットする．次の ADD によって，レジスタ A に $5 \cdot x_n$ ができる．このあと1を加え，レジスタ C に $5 \cdot x_n+1$ (mod 256) を与える．

図 7 RND8 サンプル・プログラム

レジスタ B をループ・カウンタとして使用．初期値として 0 を入れておくことによって，256 回ループをまわる．

図 8 RND8X ソース・リスト

RND8 をコールし，0~255 の値をレジスタ C に入れる．レジスタ B には区間の上限値がいれられているので，繰り返しレジスタ B を引けなくなるまで（キャリーがたつまで）減算を繰り返す．キャリーがたったなら，数は範囲内にはいったので，これを乱数とするためにレジスタ C に移す．

図 10 RND8X サンプル・プログラム

図 11 RND16 ソース・リスト

H-L レジスタにある x_n を3ステップで 257 倍する．このうち1を加え，正の整数とするために 7F で AND をとり，x_{n+1} を H-L レジスタに作成する．


```
                ;  ** RND16 SAMPLE **
                ;
                ;  AVERAGE OF 4096 RANDOM NUMBERS
                ;
382F    010010  R16TST: LXI   B,4096
3832    110000          LXI   D,0
3835    D5              PUSH  D
3836    210104          LXI   H,1025
3839    CD1739  R16T1:  CALL  RND16
383C    EB              XCHG
383D    19              DAD   D
383E    D24438          JNC   R16T2
3841    E3              XTHL
3842    23              INX   H
3843    E3              XTHL
3844    EB      R16T2:  XCHG
3845    0B              DCX   B
3846    78              MOV   A,B
3847    B1              ORA   C
3848    C23938          JNZ   R16T1
384B    E1              POP   H
384C    29              DAD   H
384D    29              DAD   H
384E    29              DAD   H
384F    29              DAD   H
3850    7A              MOV   A,D
3851    0F              RRC
3852    0F              RRC
3853    0F              RRC
3854    0F              RRC
3855    E60F            ANI   0FH
3857    B5              ORA   L
3858    6F              MOV   L,A
3859    CD1840          CALL  PUTHL
385C    C9              RET
                ;
                ;  ** RNDX ** 16BIT
                ;
3920    C5      RNDX:   PUSH  B
3921    CD1739          CALL  RND16
3924    CD2939        CALL  DIVIDE
3927    C1              POP   B
3928    C9              RET
                ;  -- DIVIDE --
                ;
                ;   (H,L)/(D,E)=(B,C)..REM IN (H,L)
                ;          (D,E) UNCHANGED ON EXIT
                ;
3929    E5      DIVIDE: PUSH  H
392A    6C              MOV   L,H
392B    2600            MVI   H,0
392D    CD3439          CALL  DIV1
3930    41              MOV   B,C
3931    7D              MOV   A,L
3932    E1              POP   H
3933    67              MOV   H,A
3934    0EFF    DIV1:   MVI   C,0FFH
3936    0C      DIV2:   INR   C
3937    7D              MOV   A,L
3938    93              SUB   E
3939    6F              MOV   L,A
393A    7C              MOV   A,H
393B    9A              SBB   D
393C    67              MOV   H,A
393D    D23639          JNC   DIV2
3940    19              DAD   D
3941    C9              RET
```

図 15 RND16 サンプル・プログラム

レジスタ B-C は，ループのカウンタである．発生する乱数の和はスタックのトップの2バイト・レジスタ D-E の順の4バイトの領域が想定され，そこへ計算されていく．H-L レジスタは，最初に初期値として 1025 が入れられ，以後は発生される乱数が保持される．H-L レジスタの内容は D-E へ加算されていくが，このときキャリーがでるなら上位2バイトを構成するスタックのトップを引き出し，1を加えておく．次にカウンタから1を引き，ゼロになったかをチェックする．4096 個の発生が終了したなら，次に総和を 4096 で割ることを行なう必要がある．4096 で割ることは，12 ビット右へシフトすることと等しい．12ビットのシフトをする代わりに取り出すべき16ビットは，スタック・トップ2バイトの下12ビットと D レジスタの上4ビットをそれぞれ抜き出して連結する．求められた平均値は，システム内部にある H-L レジスタを 16 進で印字する PUTHL ルーチンにより印刷してみる．

図 17 RNDX および DIVIDE のソース・リスト

使用ルーチン (DIVIDE) 中でレジスタ B-C がこわされるので，'PUSH B' を行なう．0~32767 のレンジの乱数を発生させる．このあと指定する区間は D-E レジスタにはいっているので，H-L レジスタを D-E レジスタで割り，その余りを求める乱数とする．

DIVIDE ルーチンは，減算を繰り返すことにより，商を B-C レジスタに，余りを H-L レジスタにおくルーチンである．

bit

図 18 RNDX サンプル・プログラム

```
                ;
                ;  ** RNDX SAMPLE **
                ;  AVERAGE OF 4096 NUMBERS
                ;
385D  010010    R16XT:  LXI   B,4096
3860  110000            LXI   D,0
3863  D5                PUSH  D
3864  D5                PUSH  D
3865  210104            LXI   H,1025
3868  111027            LXI   D,10000
386B  CD2039    R16X1:  CALL  RNDX
386E  EB                XCHG
386F  E3                XTHL
3870  19                DAD   D
3871  E3                XTHL
3872  D27C38            JNC   R16X2
3875  33                INX   SP
3876  33                INX   SP
3877  E3                XTHL
3878  23                INX   H
3879  E3                XTHL
387A  3B                DCX   SP
387B  3B                DCX   SP
387C  EB        R16X2:  XCHG
387D  0B                DCX   B
387E  78                MOV   A,B
387F  B1                ORA   C
3880  C26B38            JNZ   R16X1
3883  D1                POP   D
3884  E1                POP   H
3885  29                DAD   H
3886  29                DAD   H
3887  29                DAD   H
3888  29                DAD   H
3889  7A                MOV   A,D
388A  0F                RRC
388B  0F                RRC
388C  0F                RRC
388D  0F                RRC
388E  E60F              ANI   0FH
3890  B5                ORA   L
3891  6F                MOV   L,A
3892  CD1840            CALL  PUTHL
3895  C9                RET
```

```
                ;
                ;  ** RND ** (0,1)  REAL NUMBER
                ;
3942  E5        RND:    PUSH  H
3943  216C39            LXI   H,SAVRE
3946  CD0320            CALL  RSTOR
3949  2A7039            LHLD  INTRN
394C  CD1739            CALL  RND16
394F  227039            SHLD  INTRN
3952  CD1220            CALL  ILOAD
3955  216839            LXI   H,FMAX
3958  CD0F20            CALL  RDIV
395B  E1                POP   H
395C  E5                PUSH  H
395D  CD0320            CALL  RSTOR
3960  216C39            LXI   H,SAVRE
3963  CD0020            CALL  RLOAD
3966  E1                POP   H
3967  C9                RET
3968  FFFE000F  FMAX:   DB    0FFH,0FEH,00H,0FH
0004            SAVRE:  DS    4
3970  0104      INTRN:  DW    1025  ; INITIAL VALUE
```

図 19 RND ソース・リスト

レジスタを退避したのちに，RND16 を呼び出し，H-L レジスタに 0～32767 の整数乱数を発生させる．これを実数変換し，さらに正の最大数 32767.0 で割って区間 [0,1] の値を作り，引数としてエントリー時に H-L レジスタでわたされた 4 バイトのバッファへ転送する．レジスタを復帰して処理を終える．

図 22 RND サンプル・プログラム (PI)

レジスタ B-C を m，レジスタ D-E を i に相等するカウンタとし，図 21 をそのまま実現する．$x^2+y^2 \leq 1.0$ のチェックは，$x^2+y^2-1.0$ を計算し，結果が負であるかを見ることによって行なっている．また π は，$(m/n)\times 4$ によって計算できるが，この場合 $n=1000$ なので m を 250.0 で割ることによって求めている．

```
                ;
                ;  **RND SAMPLE **
                ;
                ;  PI NO KEISAN
                ;
3896  010000    PI:     LXI   B,0
3899  11E803            LXI   D,1000
389C  21F438    LOOP:   LXI   H,X
389F  CD4239            CALL  RND
38A2  21F838            LXI   H,Y
38A5  CD4239            CALL  RND
38A8  21F438            LXI   H,X
38AB  CD0020            CALL  RLOAD
38AE  CD0C20            CALL  RMUL
38B1  CD0320            CALL  RSTOR
38B4  21F838            LXI   H,Y
38B7  CD0020            CALL  RLOAD
38BA  CD0C20            CALL  RMUL
38BD  21F438            LXI   H,X
38C0  CD0620            CALL  RADD
38C3  21EC38            LXI   H,F1
38C6  CD0920            CALL  RSUB
38C9  21FC38            LXI   H,MEMOR
38CC  CD0320            CALL  RSTOR
38CF  3AFF38            LDA   MEMOR+3
38D2  07                RLC
38D3  D2D738            JNC   ¥+4
38D6  03                INX   B
38D7  1B                DCX   D
38D8  7A                MOV   A,D
38D9  B3                ORA   E
38DA  C29C38            JNZ   LOOP
38DD  60                MOV   H,B
38DE  69                MOV   L,C
38DF  CD1220            CALL  ILOAD
38E2  21F038            LXI   H,F250
38E5  CD0F20            CALL  RDIV
38E8  CD1B20            CALL  RPUT
38EB  C9                RET
38EC  80000001  F1:     DB    80H,00H,00H,01H
38F0  FA000008  F250:   DB    0FAH,00H,00H,08H
0004            X:      DS    4
0004            Y:      DS    4
0004            MEMOR:  DS    4
                END
```

bit