数理解析研究所講究録
第498巻 1983年 23-38

乱数パッケージの設計

慶應大・理工　　西村　和夫（Kazuo Nishimura）

擬似乱数を生成するアルゴリズムは数多く研究されているが，実際のパッケージは必ずしも知られている最良の方法を用いていない．ここでは乱数パッケージを作成するときに考慮すべき点をまとめている．近頃はソフトウェアにかかる費用が高くなってきたので，プログラムには移植性があることが重要である．特にパッケージは多くの利用者に長期にわたって使われるものなので，これから開発するものは高い移植性をもつように設計されるべきである．この論文では移植性について特に詳しく述べてある．

L.Schrage〔1〕はほとんどの計算機に移植できる，乗算合同法による一様乱数のプログラムを発表した．それに対して G. Fishman と R.Moore〔2〕は，その中で用いられている乗数がよくないと指摘している．ここでは，彼等による各種の検定の結果が良かった乗数を用いた，新しい移植性の高いプログラムを示す．すべての中間結果が $\pm(2^{31}-1)$ 以内に納まるようにしてあるので，このプログラムを整数の桁あふれを許さない言語に変換することも容易にできる．

1．設計において考慮すべき点

(a) 用途　乱数を使用する分野には，ＯＲ，統計計算，物理計算などが考えられる．ＯＲではシミュレーションに，統計ではパーセント点の計算などに乱数を用いている．パーセント点の計算では高い精度の乱数が必要である．物理では，量子力学や物性論で大量（たとえば10^6個）の乱数を用いたシミュレーションをすることがある．このような用途の違いによって必要とする乱数の精度や使用量が異なり，大量の乱数を必要とする場合には特別に設計した乱数生成プログラムが必要になる．したがって，一般の利用者を対象とするパッケージは，大量の乱数を使用する計算には向かないものになりがちである．

(b) 移植性　パッケージの移植性は，それを呼び出すプログラムの移植性に関わるので重要である．計算機の更新時などに，乱数を用いているプログラムを移植するためには，まずその基礎になっている乱数パッケージが移植できなければならない．しかし，移植性は速度と裏腹の関係にあり，乱数の生成プログラムは速度を要求されるので，移植性が犠牲になることが多いようである．

(c) 言語　パッケージで使用する言語の選択は，パッケージの利用率や移植性に大きく関わってくる．言語によっては整数の桁あふれを許さないので，乱数の生成に困ることがある．現在のところ，過去のプログラムの蓄積やプログラマの

慣れなどの理由で，FORTRAN がよく使われている．これからは徐々にFORTRAN77 に移行していくであろう．

(d) 形式　ライブラリを提供するときの形式には，大きく分けて，オブジェクト・プログラムの集合体にするのと，ソース・プログラムの集まりにするのとの二通りがある．伝統的には前者が用いられてきたが，コンパイルの費用が相対的に安くなってきており，開発環境も整ってきたので，今後は後者がもっと用いられるようになるであろう．ソース・プログラムを提供するようにすると，利用者がアルゴリズムを確認することができ，自分で都合のよいように書き直すことができるという長所があるが，一方，版権を保護する方法が問題になってくる．

(e) どのような分布を含めるか　乱数パッケージを設計するときには，各種の確率分布の内のどの分布を対象として採り入れるかを吟味しなければならない．数が多いほど良いというものではなく，全体として採択の基準がはっきりしているべきである．

(f) 生成方法の選択　採択する分布が決定したら，それらを生成するアルゴリズムを選択することになる．場合によっては，精度が高い版と，精度は低いが速い版の二通りを用意したほうが良いかもしれない．

(g) 試験　普通，パッケージを作成すると同時に，テスト・プログラムも作成する．そのテスト・プログラムやテスト

データとその結果は，整理して残しておくべきである．それらを公開すれば，利用者は安心するし，移植時に非常な助けになる．たとえば，各種の乱数の始めの数個と 1000 個目の値，2000 個目の値などを表にしておくようにするとよいであろう．何個目の値を公表するかについて規格があれば便利である．

(h) 文書　どのようなプログラムでも，仕様を文書にしておくのは重要であるが，特にパッケージの場合には，マニュアルが死命を決定するといっても過言ではない．TSS のもとでは，オンラインのマニュアルが便利である．

(i) 保守　これは設計時の当面の問題ではないが，保守をどのように行っていくかは設計時から考えておくべきである．誤りの訂正や，使い易さ等のための仕様の変更，新しいアルゴリズムを用いたプログラムの追加や置き換えなどは必ず起こることであり，これをしないパッケージは使われないものである．

2．パッケージの形式

乱数パッケージの形式には，オブジェクト・プログラムのライブラリとするのと，ソース・プログラムのライブラリにするのとの二通りが考えられる．そして，それぞれに実現の仕方でまた二通りずつが考えられる．

(a) オブジェクト・プログラムのライブラリ

① リンケージ・エディタでリンクする.

② 実行時に（ダイナミックに）リンクする.

(b) ソース・プログラムのライブラリ

① エディタ使用時に引用する.

② プリプロセッサを用いマクロで引用する.

(a)①は以前からよく行われてきた方法であり，便利な反面原理を知らずに誤った適用をする危険がある．また，この利用の仕方には，サブルーチンにするか，関数にするかの二通りがある.

(a)②はIMSLの固有値・固有ベクトル計算パッケージEISPACKにおいて用いられている方法であり，利用者は制御プログラム（“EISPAC”）だけを呼び出せばよく，実際の各サブルーチンのロードはこの制御プログラムが実行時に行う.

(b)①は本を参照するという形で行われてきた方法であり，ライブラリの一部を利用者のプログラムに組み込んで実行を速くすることができる．ソース・プログラムが公開されている例として，ＩＭＳＬ（International Mathematical and Statistical Library）がある.

(b)②を実現しているものに，日科技研のUMSがある． UMSは利用者のプログラムをプリプロセッサに通して呼出し列を生成するようになっている．いちいちプリプロセッサにかけることは利用者に負担をかけることになるので，開発環境が整っているＯＳの下でないと使いにくいものになる．近頃は

UNIX* のように便利なＯＳが増えてきたので，これからはこのタイプのパッケージも有用である．

3．生成方法

現在，知られている主な方法には，次のようなものがある．

- 一様乱数
  - 乗算合同法・かき混ぜ法
  - Ｍ系列
  - その他
- その他
  - 棄却法
  - von Neumann-Forsytheの方法（奇偶法）
  - 分布関数の逆関数を使う方法
  - alias 法（有限離散分布に限る）

乗算合同法は，一様乱数を生成するために最も多く用いられている方法であり，その性質もよく研究されている．乗算合同法（あるいは混合合同法）を使うときには，かき混ぜ法（shuffling）も併用したほうがよい．かき混ぜ法についての数学的性質はあまり分かっていないが，少なくともしないより悪くはならないと考えられており，計算時間もほとんど消費しないからである．

Ｍ系列も，生成の速度が速く，性質がよく研究されている方法である．全体的な性質は良いが，その部分的な性質はよ

＊ UNIX はベル研究所の登録商標です．

くない．

一様乱数を生成する方法は，ほぼこの二種類に限られている．これら以外の方法としては，物理的な乱数を用いる方法があるが，その性質は明らかになっていないし，再現性がないことが大きな欠点である．

一様分布以外の分布を生成する方法は，いまのところ一様乱数になんらかの変換をするものばかりである．それぞれについては５章で述べる．

４．移植性の高い一様乱数生成プログラム

L．Schrage〔１〕は，1978年に乗算合同法に基づく移植性の高い一様乱数生成プログラムを発表した．使用した言語はFORTRANであり，その仕様は次のようなものである．

まず，計算機の整変数の精度は３１ビット以上であるという仮定がある．乗算合同法の法は$2^{31}-1$にしている．すべての中間結果を$2^{31}-1$以内に納めるようにするため，乗数は2^{15}（32768）以内にしなければならない．

実際に使われている乗数は7^5（16807）である．ところで，Knuthは乗数を法の１％以上にするように勧めている．この場合，法$2^{31}-1$の１％は約21000000であり，先の値は，これよりもはるかに小さい．

この乗数16807は，P．Lewis，A．Goodman，J．Millerら〔２〕が採用したものであり，これはIBMの科学計算パッケ

ージSSP に用いられたために各所に流布した．それはシミュレーションなどの応用プログラムを実行したときに，IBM の機械と同一の結果が出ることが重要であったためであろう．現在でもこの乗数をつかっているパッケージの例として LLANDOM，APL，SIMPL／1，IMSL などがある．この乗数についての各種の検定は，G.Fishman と L.Mooreが1982年に行なっており〔3〕，その結果によればこの値はあまりよくない．

ここで，上の乗数を改良した新しい一様乱数生成プログラムを示す．生成方法はやはり乗算合同法であり，仮定と法は先のSchrage のものと同一である．しかし，計算途中の部分積の数が多いので，プログラムは幾分か面倒になり，速度も遅くなっている（図1）．乗数は $2^{31}-1$（1073741823）以内でなければならない．すべての中間結果が±（$2^{31}-1$）以内に納まるようにしてあるので，このプログラムを整数の桁あふれを許さない言語に変換することも容易である．

実際の乗数は 397204094にした．この数は G. Fishman と L. Mooreの検定で成績がよかったものであり，スペクトル検定においても，Knuth の棄却水準を満足するものである．

5．一様分布以外の乱数の生成方法

一様分布以外の擬似乱数を生成するアルゴリズムに望まれる性質は，次の5つである．

(a) アルゴリズムが簡潔であること　（これはプログラムの

```
*-----------  PORTABLE RANDOM NUMBER GENERATOR (UNIFORM)  -------------
*
      REAL FUNCTION RANDP(X)
*
*    PORTABLE RANDOM NUMBER GENERATOR USING THE RECURSION
*         X = A*X MOD M, M=2**31-1,  A=397204094
*    THE VALUE OF A IS FROM G.S.FISHMAN AND L.R.MOORE, JASA, VOL.77, 1982
*    AUTHOR:  K.NISHIMURA, DEPT. MATH, FACL. SCI. & TECH., KEIO UNIV.
*    DATE:    83-06-15
*
      INTEGER  X, XHI, XLO
      INTEGER  A, AHI, ALO
      INTEGER  LL, LLHI
      INTEGER  LH, LHHI
      INTEGER  HL, HLHI, HLLO, K, HLLOK
      INTEGER  HH, HHHI
      INTEGER  H,  L,  C
      INTEGER  B15, B16, M
      REAL     RECIPM
*           A<=2**30-1,   M=2**31-1,     B15=2**15,  B16=2**16
      DATA  A/397204094/, M/2147483647/, B15/32768/, B16/65536/
*   GET RECIPROCAL OF M
*$    IF (TEST) THEN  1-LINE  ELSE  1-LINE
*     RECIPM = 1./FLOAT(M)
      DATA  RECIPM/4.65661287E-10/
*   DECOMPOSE A TO ( AHI<15> : ALO<15> )
*$    IF (A=397204094) THEN  1-LINE  ELSE  2-LINES
      DATA  AHI,ALO/12121,23166/
*     AHI = A/B15
*     ALO = A-B15*AHI
```

図１．移植性の高い一様乱数生成プログラム（１／３）

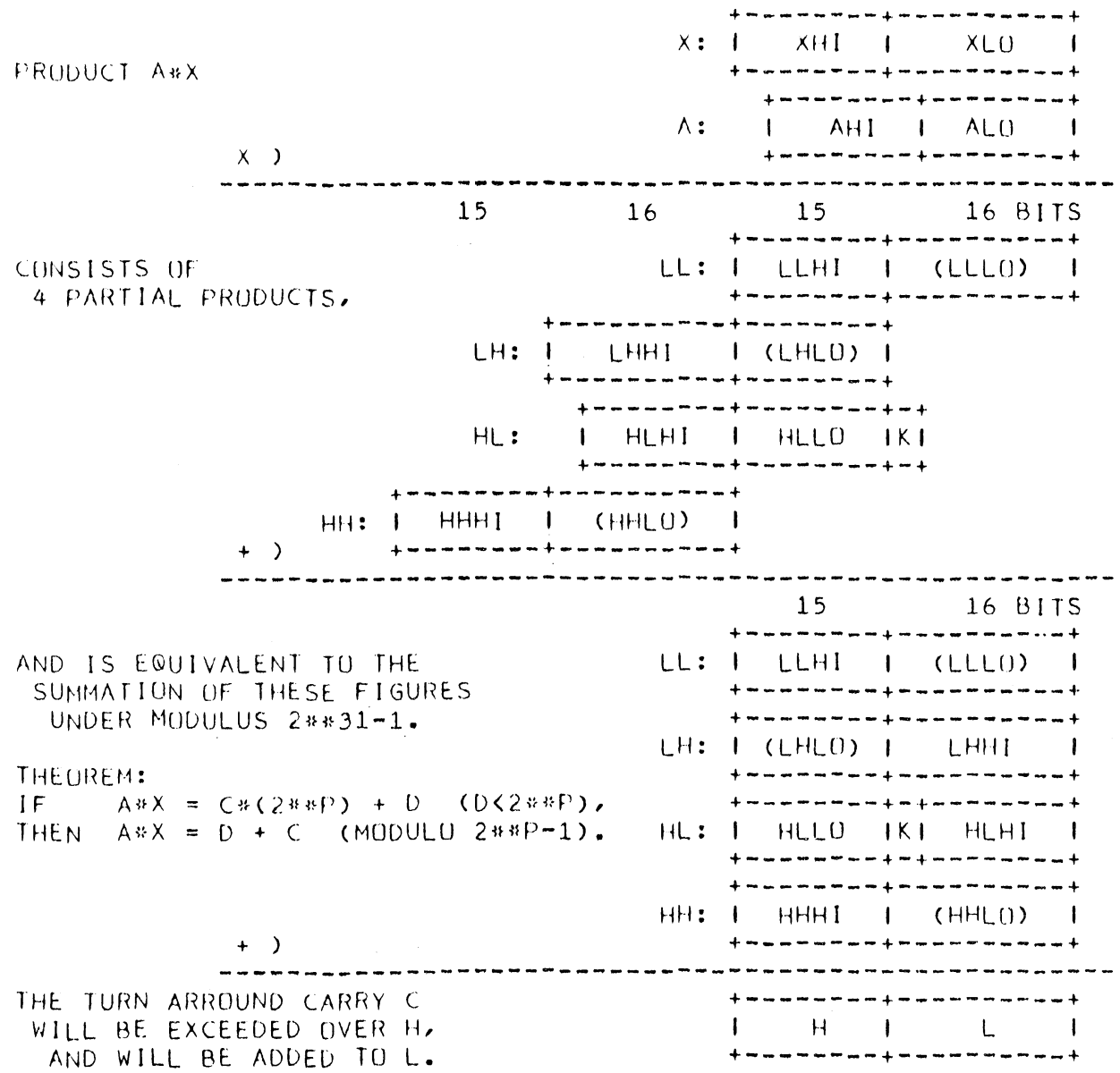

図１．移植性の高い一様乱数生成プログラム（２／３）

```
*
*   DECOMPOSE X TO ( XHI<15> : XLO<16> )
      XHI = X/B16
      XLO = X-B16*XHI
*   GET 4 PARTIAL PRODUCTS
      LL = ALO*XLO
      LH = ALO*XHI
      HL = AHI*XLO
      HH = AHI*XHI
*   GET HI ORDER 15 OR 16 BITS OF EACH PRODUCT
      LLHI = LL/B16
      LHHI = LH/B15
      HLHI = HL/B16
      HHHI = HH/B16
*   DECOMPOSE 16 LO BITS OF HL TO ( HLLO<15> : K<1> )
      HLLOK= HL-B16*HLHI
      HLLO = HLLOK/2
      K    = HLLOK-2*HLLO
*   SUM UP THE HI ORDER 15 BITS OF 4 PARTIAL PRODUCTS
      H = LLHI + (LH-B15*LHHI) + HLLO + HHHI
*   SPLIT THE TURN ARROUND CARRY C FROM H
      C = H/B15
      H = H-B15*C
*   SUM UP THE LO ORDER 16 BITS OF 4 PARTIAL PRODUCTS AND C
      L = (LL-B16*LLHI) + LHHI + B15*K + HLHI + (HH-B16*HHHI) + C
*   HERE,  0 <= H < B15,  0 <= L <= 4*B16
*   PRESUBTRACT M (=2**31-1)
      L = L - M
*   NOW, ASSEMBLE HI AND LO PARTS.  OVERFLOW NEVER OCCURS.
      X = B16*H + L
*   ADD M BACK IF NECESSARY
      IF ( X.LT.0 )  X = X + M
*   NORMALIZE X TO THE FLOATING NUMBER IN THE RANGE(0,1)
      RANDP = FLOAT(X) * RECIPM
      RETURN
      END
```

図１．移植性の高い一様乱数生成プログラム（３／３）

作成や移植性のために重要である）

(b) すべての計算が単精度で行なえること

(c) 速いこと （平均的に速いだけでなく，一つの乱数の生成時間に上限があること）

(d) 分布のパラメタが頻繁に変わってもなお十分に速いこと

(e) 結果が入力の一様乱数に対して安定していること （たとえ本の一様乱数の分布に偏りがあっても，結果の分布にひどい偏りを生じないこと）

これらの要求のいくつかは排反する．たとえば，速さを追求していくとアルゴリズムが複雑になる．上の要求のどれを優先するかは，乱数を用いるプログラムの使用目的による．

以上の要請を満足している生成方法には，以下のものがある．

棄却法（rejection method）は，John von Neumann が考案したものである．これは二つの独立な乱数を取り出して，第一の乱数を採択するかどうかを第二の乱数によって判定する方法である．生成の速度を上げるための工夫が，いろいろと考案されている．まず，判定を速く行うための搾り出し（squeeze）は，棄却法に不可欠である．棄却の回数を減らすための工夫は，渋谷が負の超幾何分布について，仁木が正規分布について行っている．G.Marsaglia の長方形楔裾分割法（ractangle-wedge-trail method）も有効である．

奇偶法（odd-even method）は，John von Neumann と G.

E. Forsythe が考案した，指数型の密度をもつ乱数を速く生成する方法である．これについても，いろいろな工夫がなされている．奇偶法では密度関数 f（x）を

$$f(x) = C \exp(-h(x))$$

と変形してから，h（x）に基づく計算をするのであるが，この変形がいつもうまくできるわけではない．いまのところ知られている適用例として，指数分布や正規分布がある．

分布関数の逆関数を用いる方法は，理解が容易であり，プログラムも簡単であるが，逆関数（あるいはそれの近似式）が陽に分かっていないと利用できない．

alias 法（同名法）は A. J. Walker が考案した，有限の離散分布を速く生成するための巧妙な方法である．有限の事象の数とその生起確率が与えられたとき，それに従う乱数を事象の数のオーダーで生成することができる．

これらのほかに，A. J. Kinderman と J.F. Monahan が考案した ratio-metod（除算法）もある．これはある領域内に生成した二つの一様乱数の比をとる方法である．片方（あるいは両方）の乱数が0に近いときは，生成される乱数の精度が低くなる．

6．生成方法の選択

ある分布に従う乱数を生成するアルゴリズムが二つ以上あるとき，それぞれの外部仕様が同じならば良いほうを採用す

るのは当然であるが，以下の場合にはパッケージに複数のプログラムを用意したほうがよい．

まず，要求される精度によって生成の速度が変わる場合である．つまり，結果の精度が高いが遅いアルゴリズムと，結果の精度が低いが速いアルゴリズムがあるときは，両方のプログラムを用意しておいたほうが便利である．

また，二項分布 N（n,p）などのようにパラメタをもつ分布の乱数を生成するとき，パラメタの変更の頻度が生成方法の選択に関わってくる．パラメタが変わらないときは表の準備ができるが，パラメタが頻繁にかわるときは表の準備ができない．したがって，選択は利用者に委ねることになる．

しかし利用者の側から見ると，ある分布に従う乱数を生成する手続き（サブルーチン）はひとつの方がよい．選択に自由度があると，利用者に本質的ではないところで負担をかけることになり，結局，利用しにくいパッケージになってしまう．

この排反する要求を解消する手段として，EISPACK や UMS のように，アルゴリズムの自動選択をする方法もある．これらについては 2 章で述べた．

7．初期値の与え方

実際のプログラムを設計するときには，一様乱数の初期値（seed）の与えかたをどうするかが問題になる．利用者の立

場からは，①常に一定の呼び出し方をしたい．②初期値を変えたい．③数種類の系列を同時に使いたい．という矛盾した要求がある．この初期値の与えかたは，使用する言語にかなり依存する．FOTRAN77を例にとって，考えられる初期値の与えかたを示すと，①引数として渡す．② ENTRY文を用いてサブルーチンに別の入口を設け，そこで設定する．③中間結果を格納する変数をCOMMON領域に登録し，これを変更する．④サブルーチン内のDATA文で設定する．などがあり得る．先程の要求をすべて満足するのは，引数として渡す方法であり，これは実際にもよく行われている．DATA文で設定してあると，ソース・プログラムが入手できないかぎり変更できない．

8．移植性

近代的なパッケージは，移植性にかなり注意をはらって設計されている．たとえばIMSLでは，プリプロセッサを用いて，言語仕様の違いや浮動小数点数の有効桁数の違いを克服している．プリプロセッサは各種の計算機に関する環境パラメタ（たとえば最大の正整数の値など）の表をもっており，プログラム内では特別な変数名を用いてそれらを参照する〔4〕．

このような環境パラメタの名称は，IFIPのWG2.5 が共通に使用できるものを提案している〔5〕．W. S. Brown と S. I. Feldman〔6〕のように環境パラメタを関数として引用することを提案している人たちもいる．それらの一部を表1に

示す．いまのところ，どの提案も世界的な標準にはなっていないが，実用性と利用者の多さの点から，筆者はIMSLのものを勧めたい．

表１．環境パラメタ（例）

IFIP-WG2.5の規定
　IOVFLO:　最大の正整数　（$2^{31}-1$など）
　SRELPR:　$1+\varepsilon \neq 1$となる最小のε（16^{-5}など）
IMSLでの変数名
　IINFP :　最大の正整数
　SEPS　:　$1+\varepsilon \neq 1$となる最小のε
W.S.Brown と S.I.Feldmanの提案
　EPEMIN（X）　:　ε
　FPFRAC（X）　:　X の少数部
　FPMAKE（X,E）:　実数の合成（$X \cdot b^E$）

最後に，この小文を発表する機会を与えて下さり，多大な御助言をして頂いた慶應義塾大学理工学部の渋谷政昭教授に深く感謝致します．

参考文献

〔１〕L.Schrage, A More Portable Fortran Random Number Genetator, ACM TOMS, Vol. 5, No. 2, 1978, pp.132-138.
〔２〕G.S.Fishman and L.R.Moore, A Statistical Evaluation of Multiplicative Congruential Random Number Generators With Modulus 2 -1, J. Amer. Stat. Asoc., Vol. 77, No. ,377, 1982, pp. 129-136.
〔３〕P. A.W. Lewis, A. S. Goodman, and J. M. Miller, A Pseudo-Random Number Generator for the System／360,IBM Systems Journal, Vol. 8, No. 2, 1969, pp.136-145.
〔４〕ソフトウェア・エンジニアリングに関する調査研究 プログラム変換系，共同システム開発，1978.
〔５〕B. Ford, Parameterization of the Environment for Transportable Numerical Software, ACM TOMS, Vol. 4, No. 2, 1978, pp. 100-103.
〔６〕W.S.Brown and S.I.Feldman, Environment Parameters for Floating-Point Computation, ACM TOMS, Vol. 6, No. 4, 1980, pp. 510-523.