01 Aug 1972

Microprogramming For Probability Distribution Sampling

Theodore Gyle Lewis
Missouri University of Science and Technology

Microprogramming for Probability Distribution Sampling

T.G. Lewis, University of Missouri – Rolla

Microprogramming of special instructions for sampling of random variates from any probability distribution is a means of increasing sampling speed. The diversity of sampling techniques is narrowed to one general algorithm; conditional bit sampling. Conditional bit sampling, uses a high speed uniform random number generator based on feedback shift registers to sample one bit at a time. The probability of a bit being a one in the j-th position of a binary expanded variate is stored in a table of conditional probabilities. A comparison with the pseudorandom number yields a one or zero. The table of conditional probabilities is generated once and passed through an instruction to the microprogram which performs the sampling. One user instruction is issued for each variate returned.

INTRODUCTION

Microprogramming, under the control of users, allows special purpose computers to be implemented through software changes (minor hardware changes to ROM)[1]. The user may design instructions particular to his application without great speed degradation [10]. A computer manufacturer may design and build machines with great flexibility and yet retain low cost.

Advances in control memory have made dynamic alteration of control memory possible [3]. The instruction repertoire may be changed under program control in order to adapt to a special purpose application. Thus, microprogramming is emerging as a design tool within reach of the applications implementor.

Non-deterministic simulation requires sampling from various probability distributions. Probabilistic simulation with a digital computer is based on pseudorandom number generators [2, 13, 4]. Most non-uniform probability distributions are sampled using uniformly random numbers [11, 12]. Software routines and special simulation languages exist for a wide range of simulation models, however, sampling speed is moderate. To increase efficiency of sampling a user may implement special hardwired computers or microprogram a general purpose computer. Approporiate techniques for microprogrammed sampling from probability distributions presented here are based on shift register theory and conditional probability tables.

UNIFORM RANDOM NUMBERS

A microprogram for generating a uniform random integer is given in figure 1 [4, 5]. Programmed on the Interdata 4, this instruction generates a uniformly random integer each time it is issued by a user [6, 7]. A memory table of length n+2 is organized in the 16-bit user memory as shown in Figure 2 and passed via address to the microprogram.

The GFSR (generalized feedback shift register) algorithm is:

1. I ← I + 1
2. IF I > n THEN I ← 1

```
                        ORG X'428'
0428  5004    RND       L   RAH,H(RND)
0429  3080              C   SB
042A  4453              L   MR4,MAR          SAVE TABL IN MR4.
042B  43A3              L   MR3,MDR          SAVE I IN MR3.
042C  5802              L   AR,X'2'
042D  C550              A   MAR,MAR,NF+NC
042E  3100              C   MR               GET S,N: LOC TABL+2.
042F  C050              A   MR0,MAR,NF+NC    SAVE TABL+4 IN MR0.
0430  58FF              L   AR,X'FF'
0431  82A3              N   MR2,MDR,NF       MASK-OFF S; N IN MR2.
0432  48AF              L   AR,MDR,CS
0433  91FF              N   MR1,X'FF'        CROSS SHIFT AND MASK-C
0434  4833              L   AR,MR3           COMPARE I:N.
0435  E824              S   AR,MR2,NC
0436  1138              B   L,OK             BRANCH IF I<N.
0437  5300              L   MR3,X'00'        OTHERWISE, SET I=0
0438  4813    OK        L   AR,MR1
0439  C130              A   MR1,MR3,NF+NC    J=I+S
043A  4813              L   AR,MR1
043B  E224              S   MR2,MR2,NC       (MR2)=J-N.
043C  113E              B   L,OK2            BRANCH IF J<N.
043D  4123              L   MR1,MR2          OTHERWISE, SET J=J-N.
043E  4818    OK2       L   AR,MR1,SL+NC     D1=2*J.
043F  C500              A   MAR,MRO,NF+NC    LOC TABL+D1+4.
0440  3100              C   MR               GET TABL(J).
0441  4838              L   AR,MR3,SL+NC     D2=2*I.
0442  C500              A   MAR,MRO,NF+NC    LOC TABL+D2+4.
0443  48A3              L   AR,MDR           SAVE TABL(J).
0444  3100              C   MR               GET TABL(I).
0445  AEA3              X   YD,MDR,NF        EX-OR TABL(J)+TABL(I).
0446  4AE3              L   MDR,YD
0447  3200              C   MW               RESULT IN TABL(I).
0448  5801              L   AR,X'01'         INCREMENT.
0449  CA30              A   MDR,MR3,NF+NC    I=I+1.
044A  4543              L   MAR,MR4
044B  3200              C   MW               NEW I IN LOC TABL.
044C  4563              L   MAR,LOC
044D  066B              D   LOC,LOC,P2N
                        END
```

Fig. 1--Interdata 4 microprogram for GFSR. Execute, RNG REG1, TABLE. The resulting random integer is returned to register REG1, and the address of TABLE locates the memory table.

3. $j \leftarrow j + S$

4. IF $j > n$ THEN $j \leftarrow j - n$

5. $W_I \leftarrow W_I \oplus W_j$.

GFSR must be initialized by setting I = 0, and generating an initial set of words, W_I, figure 2. Note that I runs from 1 to n in the algorithm, but to facilitate addressing runs from 0 to n-1 in the microprogram.

The GFSR algorithm is fast and produces arbitrarily long period sequences [4]. It is based on composite feedback shift register sequences which in turn are based on primitive trinomials over a binary field: $X^n + X^s + 1$ [4, 13]. For example, $X^4 + X^1 + 1$, yields a basic binary sequence which may be combined with other delayed basic sequences to produce a sequence of random integers, figure 3. The recurrence relation used to produce a basic sequence is $a_I = a_{I-n+s} \oplus a_{I-n}$, or $a_I = a_{I-3} \oplus a_{I-4}$, in figure 3. The parameter, n and s are

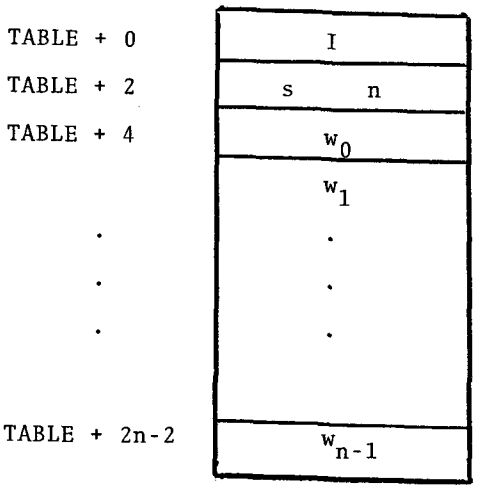

Fig. 2--Organization of TABLE for microprogram RND. GFSR uses $x^n + x^s + 1$ by exclusive-or of w_n with w_{n+s}.

selected from a list of primitive trinomials and by considering memory and period requirements [4].

Statistical properties are guaranteed when relative delay between basic sequences is great [4]. An initialization routine may be run once for a particular trinomial, $X^n + X^s + 1$, and stored in a spare table in memory. The rotating table of figure 2 is altered during use and must be reset (by copying from the spare table) if the sequence of random numbers is to be reproduced.

One instruction ($D4_{16}$ in Interdata 4) is convenient for the user and when microprogrammed becomes very fast (few memory cycles). Production of several hundred thousand numbers per second is possible with firmware.

SAMPLING FROM DISTRIBUTIONS

Distribution sampling is an art [12]. Special subroutines based on analytic inversion, rational approximation, rejection, and composition have been implemented in software. For example, to sample from $p(x) = 3X^2$, $0 \leq X \leq 1$, merely solve for X in the inversion formula; $R = X^3$. Here, R is a uniform random number, $0 \leq R \leq 1$ and X is the positive cube root of R.

Microprogramming of each special case distribution would involve large amounts of control storage and microprogramming effort. Instead, a general method applicable to all continuous distributions is implemented as a microprogram. The resulting microcoded instruction operates on a table of conditional probabilities and returns one sampled integer each time it is issued.

The conditional probability table is created by computing a ratio of areas as shown in figure 4, [8]. Briefly, the binary number to be sampled is constructed by sampling for one bit at a time to get $SX_1\ X_2\ .\ X_3\ X_4 \ldots$. The sign bit, S, and each bit, X_i i=1,2... is sampled by comparing a random number (produced by GFSR) with the conditional probability. The conditional probability table for p(x) is generated by the process given for variates with binary point between x_2 and x_3, without loss of generality.

$$\Phi(Z) = \int_0^Z p(x)\,dx \quad \text{[assume S sampled separately]}$$

$$P(x_1=1) = 2[\Phi(100.00\ldots) - \Phi(10.00\ldots)]$$

$$P(x_1=0) = 1 - P(x_1 = 1)$$

$$P(x_2=1 \mid x_1 = 0) = \frac{\Phi(01.111\ldots=10.00\ldots)-\Phi(01.00\ldots)}{\Phi(01.11\ldots=10.00\ldots)-\Phi(00.00\ldots)}$$

$$= \frac{2[\Phi(2) - \Phi(1)]}{2[\Phi(2) - \Phi(0)]}$$

Similarly,

$$P(x_2 = 1 \mid x_1 = 1) = \frac{P(x_1=1 \text{ and } x_2=1)}{P(x_1 = 1)}$$

$$= \frac{\Phi(4) - \Phi(3)}{\Phi(4) - \Phi(2)}$$

In genral, for the magnitude portion, $\bar{X}_j = X_1\ X_2\ .\ X_3 \ldots X_j$, it is possible to inductively obtain;

$$P(x_{j+1} = 1 \mid \bar{x}_j) =$$

1 1 1 1 0 0 0 1 0 0 1 1 0 1 0 : basic sequence

$a_0 = a_1 = a_2 = a_3 = 1$

$a_I = a_{I-3} \oplus a_{I-4}$; I = 4,5...

W_0	W_1 . . .												W_{14}	
1	1	1	1	0	0	0	1	0	0	1	1	0	1	0
0	0	0	1	0	0	1	1	0	1	0	1	1	1	1 : composite of delayed
0	0	1	1	0	1	0	1	1	1	1	0	0	0	1 basic sequences

W_0 = 1 0 0
W_1 = 1 0 0
W_2 = 1 0 1
1 1 1
0 0 0
0 0 1
0 1 0 : Three-bit words obtained by reading down
. 1 1 1 the columns of the composite, above.
. 0 0 1
. 0 1 1
1 0 1
1 1 0
0 1 0
1 1 0
W_{14} = 0 1 1

Fig. 3--Composite of $X^4 + X^1 + 1$ for delay of 4 bits between the columns of the three bit integer result.

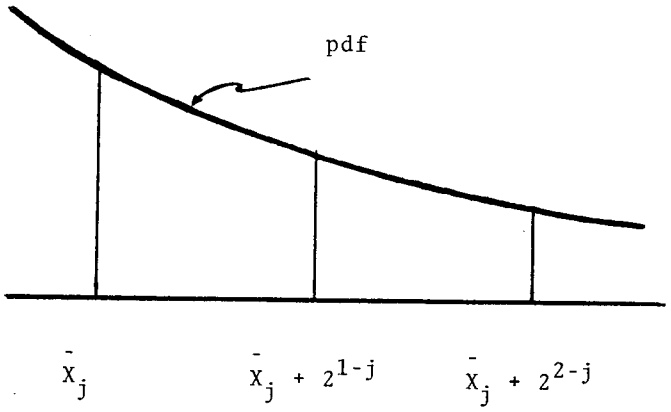

Fig. 4--Areas under consideration in computing conditional probability for pdf to be sampled.

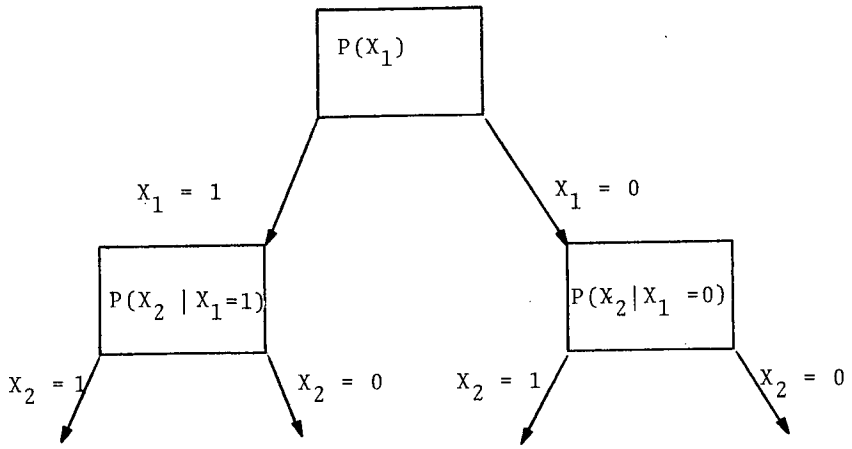

Fig. 5--A conditional probability tree for bit sampling.

$$\frac{\Phi(\bar{x}_j + 2^{2-j}) - \Phi(\bar{x}_j + 2^{1-j})}{\Phi(\bar{x}_j + 2^{2-j}) - \Phi(\bar{x}_j)} \quad ;$$

$j = 1,2...$

$P(x_1 = 1|\bar{x}_o) = P(x_1 = 1)$

Thus, a conditional probability tree as shown in figure 5 is created for bit sampling.

Substitution of $Z_j = \bar{x}_j + 2^{2-j}$ into the expression for $P(x_{j+1} = 1|\bar{x}_j)$ yields:

$$P(x_{j+1} = 1|\bar{x}_j) =$$

$$\frac{1}{2} \quad \frac{[\Phi(Z_j) - \Phi(Z_j - 2^{1-j})]/2^{1-j}}{[\Phi(Z_j) - \Phi(Z_j - 2^{2-j})]/2^{2-j}}$$

and,

$$\lim_{j\to\infty} P(x_{j+1} = 1|\bar{x}_j) = \frac{1}{2}\frac{d\Phi/dZ}{d\Phi/dZ} = \frac{1}{2}$$

Thus, if Φ is differentiable, then the occurrence of the j-th bit of X is uniformly distributed in the limit. This implies that a uniform random number may be appended to the low order bits of a random variate sampled from a differentiable distribution. The details of this technique can be found in [8], but suffice it to say that greater accuracy is obtained simply by augmenting the random variate with a random number. The number of bits to be sampled by conditional bit sampling may be determined by observing the convergence of $P(x_{j+1}|\bar{x}_j)$ to one-half. For example, for the normal distribution (u=0, σ = 1) convergence is very close to one-half after seven bits are sampled.

The binary point in $SX_1\ X_2\ .\ X_3...$ may be moved so that scaling to an integer is possible during sampling and then replaced after obtaining a sample. Also, increased accuracy is obtained by arbitrarily selecting the number of leading bits (to the left of point). Finally, S is sampled by weighing according to the fraction of samples which are negative/positive.

The total number of random numbers required is moderately high compared to specialized routines, but this is offset by the speed of GFSR. The CBS (conditional bit sampling) algorithm is:

1. I ← 0
2. J ← 0
3. M ← I + 2 ↑ J
4. I ← 2*I
5. IF RAND ≤ TABLE(M) THEN I ← I+1
6. J ← J+1
7. IF J < L GOTO 3.
8. STOP I IS SAMPLED VARIATE

To normalize I in CBS to floating point number with K bits to the left of the binary point:

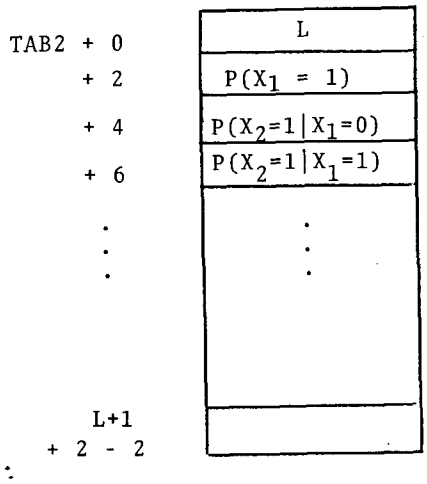

Fig. 6--Organization of CBS algorithm table, TAB2, containing conditional probabilities.

Float (i) ← I/(2↑(L-K))

The selection of L is determined by convergence of probability table, TABLE, which is linear array containing the tree of probabilities.

The CBS algorithm is easily microprogrammed as instruction, VAR REG1, TABLE, just as before in GFSR. A user-memory table, TABLE, is passed to the VAR microprogram. In microprogrammed machines which do not allow sub-microprograms to be called (transfer within control memory) a more complex instruction, VAR, REG1, TAB1, TAB2 is required:

REG1 = returned random variate

TAB1 = address of GFSR table

TAB2 = address of CBS table

Therefore, the RND micro-routine is coded as an in-line routine. The organization of TAB2 is given in figure 6.

DISCRETE DISTRIBUTION SAMPLING

Sampling from empirical or discrete variable distribution is more simplified than from the continuous. The limiting uniform distribution no longer applies, however, (not needed) and some of the conditional probabilities may even be zero.

To compute conditional probabilities, say for the Gamma distribution,

$$p(x) = \frac{x^{p-1}\, e^{-x/b}}{\Gamma(p)\, b^{p}}$$

$$p \geq 1,\ b>0,\ 0 \leq X \leq \infty$$

an integration is performed over sections of p(x). To create a conditional probability table for, say the Poisson distribution,

$$p_{\lambda}(x) = \frac{e^{-\lambda}\, \lambda^{x}}{x!}$$

$$\lambda>0,\ x = 0,1,2,\ldots$$

merely compute sums:

$$\Phi(m) = \sum_{i=o}^{m} P_{\lambda}(i)$$

The table is limited by assumming a value of m', such that;

$$\Phi(m') \doteq 1$$

The value of m' depends upon λ, but is usually small, e.g. m' = 40 for $\lambda \leq 20$. The CBS algorithm is demonstrated using Fortran routines, IPTABL and POISON.

Subroutine IPTABL figure 7 returns the one probability and 2^{L}-2 conditional probabilities in the proper order with $P(X_1=1)$ stored at dummy location TABLE(1). An example coding of IPTABL is given for the Poisson distribution where λ is passed in FL. The total number of bits to be generated by table compares is passed in dummy argument L. ERR=0 indicates normal termination of IPTABL while ERR=-1 signals an error. The value in TABLE where the error occurred is also replaced by -1.

Probabilities between arbitrary limits are computed with function subprogram POISON(Poisson). Minus one is returned for unacceptable arguments figure 8.

SUMMARY

Two special purpose instructions microprogrammed to perform the GFSR and CBS algorithms result in a wide variety of sampling possibilites merely by providing tables. The tables are generated once for each distribution required and remain unchanged (read-only). Greater speed is obtained, however, if the tables are set in control memory, since for a L bit CBS sample, L + 1 memory fetches are required from the conditional probability table and 2L + 2 fetches and L +1 stores are required in the GFSR routine. Altogether 4(L + 1)

```
      SUBROUTINE IPTABL(L,TABLE,ERR,FL)
C L=TOTAL NUMBER OF BITS SAMPLED USING TABLE.
C J2=2**J IN ITERATIVE FORM, USED TO INDEX TABLE.
C MM=2**(L-J) IN ITERATIVE FORM.
C JC=BIT COUNTER, COUNTS TO L.
      DIMENSION TABLE(1)
      ERR=0
      MM=2**(L+1)
      J2=1
      DO 1 JJ=1,L
      JC=JJ-1
      MM=MM/2
C II=PERMUTATION COUNTER.
      DO 2 II=1,J2
C OBTAIN BIT COMBINATION IN INTEGER FORM.
      I=II-1
C NORMALIZE BIT COMBINATION TO GET IXBAR(J).
      IXBAR=I*MM
C CALCULATE UPPER AND LOWER LIMITS.
      IHI=IXBAR+MM-1
      ILOW=IXBAR+MM/2
      TOP=POISON(ILOW,IHI,FL)
      BOT=POISON(IXBAR,IHI,FL)
      IF(TOP.LT.0.OR.BOT.LT.0) GO TO 4
C CALCULATE TABLE ENTRY.
      IF(BOT.GT.0) GO TO 3
      TABLE(I+J2)=0
      GO TO 2
4     TABLE(I+J2)=-1.
      ERR=-1.
      GO TO 2
3     TABLE(I+J2)=TOP/BOT
2     CONTINUE
      J2=2*J2
1     CONTINUE
      RETURN
      END
```

Fig. 7--FORTRAN subroutine for computing conditional probability table of CBS algorithm and POISSON distribution.

```
      FUNCTION POISON(ILOWER,IUPPER,FL)
C POISON = SUM ON I FROM ILOWER TO IUPPER OF
C EXP(-FL)*FL**I/I FACTORIAL.
      IF(FL.LE.0.OR.ILOWER.LT.0.OR.
     XIUPPER.LT.0) GO TO 3
      IF(ILOWER.GT.IUPPER) GO TO 1
      IUP=IUPPER+1
      ILOW=ILOWER+1
      PX=EXP(-FL)
      SUM=0
      DO 2 I=1,IUP
      IF(I.EQ.ILOW) SUB=SUM
      SUM=SUM+PX
2     PX=PX*FL/I
      POISON=SUM-SUB
      RETURN
1     POISON=0
      RETURN
3     POISON=-1.
      RETURN
      END
```

Fig. 8--FORTRAN function for computing POISSON cumulative distribution function, $\phi(m)$.

memory references are required compared to half that many when control memory is used for TAB2. Whether or not control memory is used for table storage depends upon the machine's architecture and memory space.

Higher level simulation languages are appropriate for use with these microprogrammed instructions. For example, a simulation language (SL) compiler easily generates the corresponding conditional probability table and issues a VAR instrucion. The comiler may even determine table length, L, by observing convergence to $P(X_{j+1}|X_j) \doteq 0.5$. Greater resolution may be obtained by augmentation with a uniform random number from RND.

Dynamic vector allocation would allow creation and deletion of probability tables as needed. Storage is conserved but the tables may have to be generated again. Details and programs for generating CBS tables are given in reference 8.

The CBS algorithm is completely general, and because of the speed of GFSR and control memory, highly desirable. Software sampling routines may require far fewer random numbers, but this becomes less improtant with rapid means of producing random numbers. Microprogramming of present software routines is impractical because of their diversity. Thus, a general algorithm which fits a wide class of sampling problems is ideal for microprogramming.

Finally, little has been said about the arbitrary sense of CBS, but it should be noted that any distribution that can be generated by a digital computer can be generated by the microprogrammed CBS algorithm. Therefore, empirical or experimental distributions, represented by histograms, may also be applied merely by supplying a conditional probability table.

REFERENCES

1. Flynn, M.J. and Rosin, R.F., "Microprogramming: An Introduction and a Viewpoint", IEEE Trans Comp. C-20,7, (July 1971)
2. Hull, T.E. and Dobell, A.R., "Random Number Generator", SIAM Review, 4, (1962) 230-254.
3. Lawson, H.W. and Smith, B.K., "Functional Characteristics of a Multilinqual Processor", IEEE Trans. Comp. C-20,7 (July 1971).
4. Lewis, T.G., Pseudorandom Number Generators in n-space, Ph.D. Thesis, Washington State University, 1971.
5. Lewis, T.G. and Payne, W.H., "Generalized Feedback Shift Register Random Number Algorithm," submitted to Journal ACM.
6. Micro-code Programming of a Read-Only Memory Computer, Interdata application No. 103 and No. 38-020.
7. Model 4 Micro-Instruction Reference Manual, Interdata Pub. No. 29-032.
8. Payne, W.H. and Lewis, T.G., "Conditional Bit Sampling: Accuracy and Speed," Mathematical Software, E. John Rice, Academic Press, 1971.
9. Ramamoorthy, C.V. and M Tsuchiya, "A Study of User-Microprogrammable Computers," AFIPS Conference, Proc., 36 (1970-SJCC), Montvale, N.J. AFIPS Press, 1970, pp. 165-181.
10. Shriver, B.D., "Microprogramming and Numerical Analysis," IEEE Trans. Comp. C-20,7, (July 1971).
11. Teichroew, D., "Distribution Sampling With HIgh Speed Computers," Ph.D. Thesis, North Carolina State College, 1953.
12. Tocher, K.D. The Art of Simulation, The English University Press, London, 1963.
13. Whittlesey, J.R.B. " A Comparison of the Correlational Behavior of Random Number Generators for the IBM 360", Comm. ACM. 11 (Sept. 1968), 641-644.