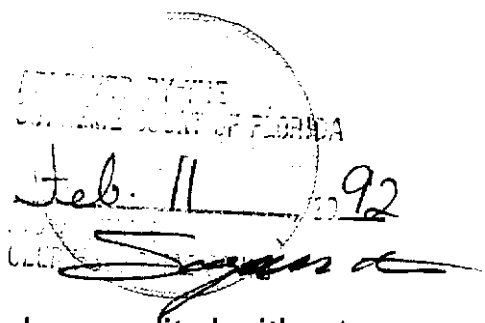
LEON COUNTY
LOCAL RULE # 92-2



WHEREAS, the present method of selecting jurors can be expedited without additional expense or loss of the sanctity of random selection by the use of the electronic computer available for use by Leon County, and

WHEREAS, the source of such selection is from the voter registration list which is in computer compatible form and under the custody and control of the Clerk of the Circuit Court of Leon County, it is therefore,

RESOLVED that the Rules of the Second Judicial Circuit for procedure in all courts of Leon County in which jury trials are held shall be amended to include this additional Rule adopting the following alternative plan for the selection of persons for grand jury or petit jury service:

1. EQUIPMENT:

(a) The equipment used in jury selection is a Memorex-Telex 7065 computer located in the secured computer room of the Clerk of Circuit Court of Leon County.

2. ALTERNATIVE METHOD OF SELECTING VENIRE:

(a) The source from which names shall be taken is the voter registration list of all precincts in Leon County, provided and certified by the Supervisor of Elections of Leon County. In every year hereafter, by the first week of January or as soon thereafter as practicable, the Supervisor of Elections will deliver and certify to the Clerk of Circuit Court a computerized listing of the current voter registration list

of all registered voters in Leon County. The Clerk of Circuit Court will protect the listing and tapes from further writing and keep it securely stored.

(b) The Clerk of Circuit Court of Leon County is designated the official custodian of the computer records of the lists to be used in jury selection and shall ensure they are not accessible to anyone other than those directly involved in selection of venires, as herein provided. Functions of the Clerk of Circuit Court may be performed by his deputies.

(c) The entire list of registered voters may comprise the certified master jury list from which venires will be selected according to the provisions of Section 2(d). Alternatively, the Chief Judge or his designated representative, with the aid and assistance of the Clerk of the Circuit Court, may select the certified master jury list for the year by lot and at random from the entire list of registered voters using the method described in Attachment A.

(d) The Clerk of Circuit Court shall cause jury venires to be selected from the final certified jury list programmed into the Leon County computer using the method described in Attachment A in accordance with directions received from the Chief Judge or his designated representative.

STATE OF FLORIDA

COUNTY OF LEON

CERTIFICATE

I hereby certify that the majority of the judges authorized to conduct jury trials in Leon County, Florida, have consented to the use of the electronic system, which is described in attachment hereto, and requests approval for the use of such system in Leon County, Florida, by the Supreme Court of Florida, as required by Chapter 40.225, Florida Statutes.

DATED this _____ day of January, 1992

CHIEF JUDGE
SECOND JUDICIAL CIRCUIT

ATTACHMENT A

Toward A Universal Random Number Generator

This article describes an approach toward a random number generator that passes all of the stringent tests for randomness we have put to it, and that is able to produce exactly the same sequence of uniform random variables in a wide variety of computers, ranging from TRS80, Apple, Macintosh, Commodore, Kaypro, IBM PC, AT, PC and AT clones, Vax, IBM 360/370, 3090, Amdahl and CDC Cyber to 205 and ETA supercomputers.

An essential property of a random number generator is that it produces a satisfactorily random sequence of numbers. Increasingly sophisticated uses have raised questions about the suitability of many of the commonly available generators—see, for example, reference [1]. Another shortcoming in many, indeed most, random number generators is the ability to produce the same sequence of variables in a wide variety of computers, an essential requirement for an experimental science that lacks standardized equipment for verifying results.

We address these deficencies here, suggesting a combination generator tailored particularly for reproducibility in all CPU's with at least 16 bit integer arithmetic. The random numbers themselves are reals with 24-bit fractions, uniform on $(0, 1)$. We provide a suggested Fortran implementation of this "universal" generator, together with suggested sample output with which one may verify that a particular computer produces exactly the same bit patterns as the computers enumerated above. The Fortran code is so straightforward that versions may be readily written for other languages; so far, students have written and confirmed results for Basic, Pascal and Modula II versions. I have not tried the Pascal version myself, as I use with reluctance a language so poorly designed that the most frequently used symbol takes three keystrokes.

A list of desirable properties for a random number generator might include:

(1 *Randomness.* Provides a sequence of independent uniform random variables suitable for all reasonable applications. In particular, passes all the latest tests for randomness and independence.

(2 *Long Period.* Able to produce, without repeating the initial sequence, all of the random variables for the huge samples that current computer speeds make possible.

(3 *Efficiency.* Execution is rapid, with modest memory requirements.

(4 *Repeatability.* Initial conditions (seed values) completely determine the resulting sequence of random variables.

(5 *Portability.* Identical sequences of random variables may be produced in a wide variety of computers, for given starting values.

(6 *Homogeneity.* All subsets of bits of the numbers must be random, from the most- to the least-significant bits.

Choice of the Method

Our choice of a generator that goes to meet these criteria is a combination generator, in which the principal, long period, component is based on the binary operation $x * y$ on reals x and y defined by

$$x * y = \{\text{if } x \geq y \text{ then } x - y, \text{ else } x - y + 1\}$$

We require a sequence of reals on $(0, 1)$: $U_1, U_2, U_3, \ldots$, each with a 24-bit fraction. We chose 24 bits because it is the most common fraction size for single-precision reals and because the operation $x * y$ can be carried out exactly, with no loss of bits, in most computers—those with reals having fractions of 24 or more bits.

This choice allows us to use a lagged-Fibonacci generator, designated F(r,s,*), as the basic component of our universal generator, providing a sequence of reals by means of the operation $x * y$:

$$x_1, x_2, x_3, \ldots \qquad \text{with } x_n = x_{n-r} * x_{n-s}.$$

The lags r and s are chosen so that the sequence is satisfactorily random and has a very long period. If the initial, seed values, $x_1, x_2, \ldots, x_r$ are each 24-bit-fractions, $x_i = I_i/2^{24}$, then the resulting sequence, generated by $x_n = x_{n-r} * x_{n-s}$, will produce a sequence with period and structure identical to that of the corresponding sequence of integers

$$I_1, I_2, I_3, \ldots \qquad \text{with } I_n = I_{n-r} - I_{n-s} \bmod 2^{24}.$$

24-bit fraction, but that seems too great a burden. After considerable experimentation, we recommend the following procedure: assign values bit-by-bit to the initial table $U(1), U(2), \ldots, U(97)$ with a sequence of bits $b_1, b_2, b_3, \ldots$. Thus $U(1) = .b_1 b_2 \ldots b_{24}$, $U(2) = .b_{25} b_{26} \ldots b_{48}$ and so on. The sequence of bits is generated by combining two different generators, each suitable for exact implementation in any computer: one a 3-lag Fibonacci generator, the other an ordinary congruential generator for modulus 169.

The two sequences that are combined to produce bits $b_1, b_2, b_3, \ldots$ are:

$$y_1, y_2, y_3, y_4, \ldots \quad \text{with} \quad y_n = y_{n-3} \times y_{n-2} \times y_{n-1} \bmod 179.$$
$$z_1, z_2, z_3, z_4, \ldots \quad \text{with} \quad z_n = 53 z_{n-1} + 1 \bmod 169.$$

Then b_i in the sequence of bits is formed as the sixth bit of the product $y_i z_i$, using operations which may be carried out in most programming languages: $b_i = \{\text{if } y_i z_i \bmod 64 < 32 \text{ then } 0, \text{ else } 1\}$.

Choosing the small moduli 179 and 169 ensures that arithmetic will be exact in all computers, after which combining the two generators by multiplication and bit extraction stays within the range of 16-bit integer arithmetic. The result is a sequence of bits that passes extensive tests for randomness, and thus seems well suited for initializing a universal generator.

The user's burden is reduced to providing three seed values for the 3-lag Fibonacci sequence, and one seed value for the congruential sequence $z_n = 53 z_{n-1} \bmod 169$. For Fortran implementations of the universal generator, we recommend that a table U(1),...,U(97) be shared, in (labelled) COMMON, with a setup routine, say RSTART(I,J,K,L), and the function subprogram, UNI(), that returns the required uniform variate. An alternative approach is to have a single subprogram that includes an entry for the setup procedure, but not all Fortran compilers allow multiple entries to a subprogram. The initial, seed values for the setup are I,J,K, and L. Here I,J,K must be in the range 1 to 178, and not all 1, while L may be any integer from 0 to 168. If (positive) integer values are assigned to I,J,K,L outside the specified ranges, the generator will still be satisfactory, but may not produce exactly the same bit patterns in different computers, because of uncertainties when integer operations involve more than 15 bits.

To use the generator, one must first CALL RSTART(I,J,K,L) to set up the table in labelled common, then get subsequent uniform random variables by using UNI() in an expression—as, for example, in X=UNI() or Y=2.*UNI()-ALOG(UNI()), etc.

FORTRAN SUBPROGRAMS FOR INITIALIZING AND CALLING UNI

```
      SUBROUTINE RSTART(I,J,K,L)
      REAL U(97)
      COMMON /SET1/ U,C,CD,CM
      DO 2 II=1,97
      S=0.
      T=.5
          DO 3 JJ=1,24
          M=MOD(MOD(I*J,179)*K,179)
          I=J
          J=K
          K=M
          L=MOD(53*L+1,169)
          IF(MOD(L*M,64) .GE. 32) S=S+T
    3     T=.5*T
    2 U(II)=S
      C=362436./16777216.
      CD=7654321./16777216.
      CM=16777213./16777216.
      RETURN
      END
```

```
      FUNCTION UNI()
      REAL U(97)
      COMMON /SET1/ U,C,CD,CM
      DATA I,J/97,33/
      UNI=U(I)-U(J)
      IF(UNI .LT. 0.)  UNI=UNI+1.
      U(I)=UNI
      I=I -1
      IF(I .EQ. 0) I=97
      J=J-1
      IF(J .EQ. 0) J=97
      C=C-CD
      IF(C .LT. 0.)  C=C+CM
      UNI=UNI-C
      IF(UNI .LT. 0.)  UNI=UNI+1.
      RETURN
      END
```

Verifying the Universality

We now suggest a short Fortran program for verifying that the universal generator will produce exactly the same 24-bit reals that other computers produce. Conversion to an equivalent Basic, Pascal or other program should be transparent. Assume then that you have implemented the UNI routine with its RSTART setup procedure in your computer. Running this short program or an equivalent:

```
      CALL RSTART(12,34,56,78)
      DO 2 I=1,20000
2     X=UNI()
      PRINT 3,(4096.*(4096.*UNI()),I=1,6)
3     FORMAT(6F12.1)
      END
```

should produce this output:

```
   6533892.0   14220222.0    7275067.0    6172232.0    8354498.0   10833180.0
```

If it does, you will almost certainly have a universal random number generator that passes all the standard tests, and all the latest—more stringent—tests for randomness, has an incredibly long period, about 2^{144}, and, for given RSTART values I,J,K,L, produces the same sequence of 24-bit reals as do almost all other commonly-used computers.

Good Luck.

References

[1] George Marsaglia, "A current view of random number generators", Keynote Address, Computer Science and Statistics: Sixteenth Symposium on the Interface, Atlanta, March 1984. In *Proceedings of the Symposium*, Elsevier, 1986.

[2] George Marsaglia and Liang-Huei Tsay, "Matrices and the Structure of Random Number Sequences", *Linear Algebra and its Applications*, 67, 147-156, 1985.

George Marsaglia
Supercomputer Computations Research Institute
Florida State University
Tallahassee, Florida