IN THE TWENTIETH JUDICIAL CIRCUIT IN AND FOR THE STATE OF FLORIDA

LOCAL RULE #VII

IN RE: SELECTION OF JURORS BY COMPUTER IN GLADES COUNTY

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 an electronic computer available for use by Glades County, and

WHEREAS, in accordance with Florida Statue § 40.011, the source of such selection is from the database of names from the Department of Highway Safety and Motor Vehicles which is in a computer compatible form and in the custody of the Clerk of the Circuit Court, and

WHEREAS, in accordance with Florida Statue § 40.011, the source of selection is also from the list of those whose names do not appear on the Department database, but who have filed with the Clerk of the Circuit Court an affidavit prescribed in the cited statue, it is therefore,

RESOLVED that the Rules of the Twentieth Judicial Circuit for procedure in all courts of Glades 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 or petit jury service:

1. EQUIPMENT:

   a. The equipment used in jury selection is Compaq Proliant 370 computer running Unixware 7.1 located in the secured computer room of the Clerk of the Circuit Court for Glades County.

APPROVED BY THE
SUPREME COURT OF FLORIDA

December 13, 20 01
THOMAS D. HALL
CLERK SUPREME COURT

2. ALTERNATIVE METHOD OF SELECTING VENIRE:

a. The source from which names shall be taken is the same as that which is described above in accordance with Florida Statue § 40.011. In every year hereafter, by the first week of January, or as soon thereafter as practicable, the Clerk of the Circuit Court shall obtain a computerized listing of names from the Department of Highway Safety and Motor Vehicles. The Clerk of the Circuit Court will protect the listing and tapes and keep them securely stored.

b. The Clerk of Circuit Court of Glades 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 the Circuit Court may be performed by his deputies.

c. The entire list of drivers license holders, identification card holders, and those who have filed affidavits pursuant to Florida Statue § 40.011 (hereinafter "eligible jurors") may comprise the master jury list from which venires will be selected according to the provisions of paragraph 2(d) below. Alternatively, the Chief Judge or his designated representative, with the aid and assistance of the Clerk of the Circuit Court, may select the master jury list for the year by lot and at random from the entire list of eligible jurors using the method described in Attachment "A" and its addendum.
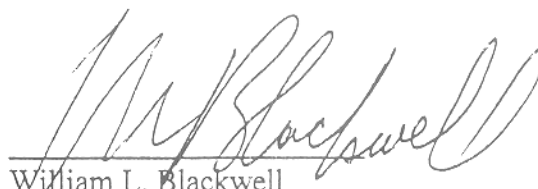
2

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

e. The initial jury selection programming may exclude persons who have been previously excluded for physical infirmities or inability to comply with other non-correctable statutory qualifications.

IN RE: SELECTION OF JURORS BY COMPUTER IN GLADES COUNTY

STATE OF FLORIDA

COUNTY OF GLADES

C E R T I F I C A T E

I HEREBY certify that, pursuant to Florida Statue § 40.225, a majority of judges authorized to conduct jury trials in Glades County, Florida, has consented to the use of the electronic system which is described in an attachment hereto, and requests the approval of the Supreme Court of Florida for the use of such system in Glades County, Florida.

DATED the 16 day of _____ Nov, _____, 2001

William L. Blackwell
Chief Circuit Judge
Twentieth Judicial Circuit

4

Attachment "A"

Prospective jurors are selected from the Florida Department of Highway Safety and Motor Vehicles drivers license file by using randomizing software by Dr. G. Marsaglia of Florida State University, and programmed by Software Systems Design Incorporated, as is described in the article entitled "Toward A Universal Random Number Generator," a copy of which is appended to this attachment.

The steps for selection are:

1.   A sequential number is attached to every eligible Glades County licensed driver file;

2.   Four seed numbers are used to randomly select the sequential number of those jurors who will form the needed venire.

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 deficiencies 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 $x \bullet y$ on reals x and y defined by

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

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 $x \bullet y$ 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 $F(r,s,\bullet)$, as the basic component of our universal generator, providing a sequence of reals by means of the operation $x \bullet y$:

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

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 $x_n = x_{n-r} \bullet x_{n-s}$, will produce a sequence with period and structure identical to that of the corresponding sequence of integers

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

For suitable choices of the lags r and s the period of the sequence is $(2^{24} - 1) \times 2^{r-1}$. The need to oose r large for long period and randomness must be balanced with the resulting memory costs: a table of the r most recent z values must be stored. We have chosen $r = 97, s = 33$. The resulting cost of 97 storage locations for the circular list needed to implement the generator seems reasonable. A few hundred memory locations more or less is no longer the problem it used to be. The period of the resulting generator is $(2^{24} - 1) \times 2^{96}$, about 2^{120}, which we boost to 2^{144} by the other part of the combination generator, described below. Methods for establishing periods for $F(r, s, \cdot \bmod 2^k)$ generators are given in reference [2].

The Second Part of the Combination

We now turn to choice of a generator to combine with the $F(97, 33, \cdot)$ chosen above. We are not content with that generator alone, even though it has has an extremely long period and appears to be suitably random from the stringent tests we have applied to it. But it fails one of the tests: the Birthday-Spacings Test. This test goes as follows: let each of the generated values $z_1, z_2, \ldots$ represent a 'birthday' in a 'year' of 2^{24} days. Choose, say, $m = 512$ birthdays, $z_1, z_2, \ldots, z_m$. Sort these to get $z_{(1)} \leq z_{(2)} \leq \cdots \leq z_{(m)}$. Form spacings $y_1 = z_{(1)}, y_2 = z_{(2)} - z_{(1)}, y_3 = z_{(3)} - z_{(2)}, \ldots, y_m = z_{(m)} - z_{(m-1)}$. Sort the spacings, getting $y_{(1)} \leq y_{(2)} \leq \cdots \leq y_{(m)}$. The test statistic is J, the number of duplicate values in the sorted spacings. i.e., initialize $J \leftarrow 0$ then for $i = 2$ to m, put $J \leftarrow J + 1$ if $y_{(i)} = y_{(i-1)}$. The resulting J should have a Poisson distribution with mean $\lambda = m^3/(4n) = m^3/2^{26}$. Lagged-Fibonacci generators $F(r, s, \cdot)$ fail this test, unless the lag r is more than 500 or the binary operation $\cdot$ is, say, multiplication for odd integers $\bmod 2^k$.

In order to get a generator that passes all the stringent tests we have applied, we have resorted to combining the $F(97, 33, \cdot)$ generator with a second generator. Combining different generators has strong theoretical support; see [1]. Our choice of the second generator is a simple arithmetic sequence for the prime modulus $2^{24} - 3 = 16777213$. For an initial integer I, subsequent integers are $I - k, I - 2k, I - 3k, \ldots$ $\bmod 16777216$. This may be implemented in 24-bit reals, again with no bits lost, by letting the initial value be, say $c = 362436/16666216$, then forming successive 24-bit reals by the operation $c \circ d$, defined as $d = \{$if $c \geq d$ then $c - d$, else $c - d + 16777213/16777216\}$. Here d is some convenient 24-bit rational, say $d = 7654321/16777216$. The resulting sequence has period $2^{24} - 3$, and while it is far too regular for use alone, it serves, when combined by means of the $\cdot$ operation with the $F(97, 33, \cdot)$ sequence, to provide a composite sequence that meets all of the criteria mentioned in the introduction, except for simplicity. All of the operations in the combination generator are simple, and the generation part is quite simple, but the setup procedure, setting the initial $97z$ values, is more complicated than the generating procedure. We now turn to details of implementation.

Implementation

We have two binary operations, each able to produce exact arithmetic on reals with 24-bit fractions:

$$z \cdot y = \{\text{if } z \geq y \text{ then } z - y, \text{ else } z - y + 1\} \qquad c \circ d = \{\text{if } c \geq d \text{ then } c - d, \text{ else } c - d + 16777213/16777216\}.$$

We require computer instructions that will generate two sequences:

$$z_1, z_2, z_3, \ldots, z_{97}, z_{98}, \ldots \qquad \text{with } z_n = z_{n-97} \cdot z_{n-33},$$
$$c_1, c_2, c_3, \ldots \qquad \text{with } c_n = c_{n-1} \circ (7654321/16777216).$$

then produce the combined sequence

$$U_1, U_2, U_3, \ldots \qquad \text{with } U_n = z_n \cdot c_n.$$

The c sequence requires only one initial value, which we arbitrarily set to $c_1 = 362436/16777216$. The z sequence requires 97 initial, seed, values, each a real of the form $I/16777216$, with $0 \leq I \leq 16777215$. The main problem in implementing the universal generator is in finding a suitable way to set the 97 initial values, a way that is both random and consistent from one computer to another. The $F(97, 33, \cdot \bmod 1)$ generator is quite robust, in that it gives good results even for bad initial values. Nonetheless, we feel that the initial table should itself be filled by means of a good generator, one that need not be fast because it is used only for the setup. Of course, we might ask that the user provide 97 seed values, each with an exact

2

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: $b_i = \{$if $y_i z_i \bmod 64 < 32$ then 0, 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 $U(1), \ldots, U(97)$ be shared, in (labelled) COMMON, with a setup routine, say RSTART(I,J,X,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,X, and L. Here I,J,X 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,X,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,X,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,X,L)
      REAL U(97)
      COMMON /SET1/ U,C,CD,CM
      DO 2 II=1,97
      S=0.
      T=.5
         DO 3 JJ=1,24
         X=MOD(MOD(I*J,179)*X,179)
         I=J
         J=X
         X=X
         L=MOD(53*L+1,169)
         IF(MOD(L*X,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   8354458.0   10633180.0
```

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