Uniform Pseudo-random Number Generation in *k* Dimensions using the ACORN Algorithm

Roy S Wikramaratna, RPS Energy

WikramaratnaR@rpsgroup.com

OCCAM, University of Oxford

20th April 2009

Acknowledgements

- Chris Farmer
  - Suggesting problem (1984)
  - Continuing interest in progress and results
- RPS Energy
  - Current employer
  - For giving me the time to visit today
- Numerical Algorithms Group Ltd, Oxford
  - For challenging me to provide more conclusive demonstrations of effectiveness of ACORN algorithm
  - Special thanks to Brian Ford, Martyn Byng at NAG

Outline

- Background
  - Why pseudo-random number generation
  - Desirable properties
  - Some alternative approaches to the problem
- ACORN algorithm
  - Specification; implementation; mathematical and numerical analysis
- Leading to the conclusion that ACORN algorithm is practical approach to uniform pseudo-random number generation
  - Easy to implement
  - Scales to any size of problem
    - uniformity in *k*-dimensions, any given *k*; period length in excess of any given number
  - Some very interesting analysis and useful mathematical results

Some background

- What is a *pseudo-random sequence* of numbers?
  - Sequence generated from specified algorithm and initial state
  - Algorithm chosen so that sequence *appears* random
    - Difficult to identify current state precisely without exact knowledge of the sequence
    - Small perturbations in current state make large difference to future evolution
- Many different mathematical and numerical problems whose numerical solution requires a reliable source of uniformly distributed (pseudo-)random numbers
  - Monte Carlo methods, with applications including
    - numerical integration
    - numerical optimisation
    - Bayesian inference
    - geostatistical simulation, statistical physics, other statistical applications
  - Games of chance (computer simulation of shuffling cards, dice, roulette wheels, etc)
  - Cryptography and related applications

k-distributed sequences - definitions

- A sequence ($\mathbf{x}_n$) is well distributed modulo 1 in $\Re^k$ if uniformly in p and for all [**a**,**b**) contained in or equal to $\mathfrak{I}^k$

$$\lim_{N\to\infty}\frac{A([a,b);N,p))}{N}=\prod_{j=1}^{k}(b_j-a_j)$$

- $A([\mathbf{a},\mathbf{b});N,p)$ denotes number of points $\{\mathbf{x}_{p+n}\}$, $1\leq n<N$ that lie in [**a**,**b**) where $\{\mathbf{x}\}$ means fractional parts of **x**
- A sequence ($\mathbf{x}_n$) is uniformly distributed modulo 1 in $\Re^k$ if the above expression holds for p=0
- See Kuipers and Niederreiter, Uniform Distribution of Sequences

Uniformly distributed, *k* dimensions

- K&N, **Theorem 6.4**:

  A sequence (x_n) is u.d. mod 1 in $\mathfrak{R}^k$ if and only if for every continuous complex-valued function *f* on $\mathfrak{I}^k$ the following relation holds:

$$\lim_{n\to\infty}\frac{1}{N}\sum_{n=1}^{N} f(\{\mathbf{x}_n\}) = \int_{\mathfrak{I}^k} f(\mathbf{x})\,d\mathbf{x}$$

Well distributed, *k* dimensions

- K&N, **Theorem 6.4** generalises to

  A sequence (x_n) is w.d. mod 1 in $\mathfrak{R}^s$ if and only if, uniformly in p and for every continuous complex-valued function f on $\mathfrak{I}^s$, the following relation holds:

$$\lim_{n\to\infty}\frac{1}{N}\sum_{n=1+p}^{N+p} f(\{\mathbf{x}_n\}) = \int_{\mathfrak{I}^s} f(\mathbf{x})\,d\mathbf{x}$$

Condition for convergence of Monte-Carlo integration in *k*-dimensions

- A consequence of these Theorems is that having access to sequences that approximate to being w.d. mod 1 in k dimensions (or at the very least u.d. mod 1 in k dimensions) is a requirement for successful k-dimensional Monte-Carlo integration.
- This suggests that such sequences could be of enormous potential value.
- BUT, in practice, “uniformly distributed” pseudo-random sequences rarely give a good approximation to u.d. mod 1 in more than a small number of dimensions.
- To date, ACORN sequences are the only ones for which we can prove uniformity in k dimensions for any given value k

Motivation (historical)

- Circa 1984, at Winfrith (with Chris Farmer)
  - Developing numerical applications (in particular moving point methods for convection-diffusion problems) which required uniform 'random' distribution of points in 2D (and ultimately 3D) grid cells
  - Desire for independence from commercial software and freedom to run on any machine
  - Seeking method that was simple to implement as well as reliable
- Problems and pitfalls
  - Turned out to be a bit more complicated (and a whole lot more interesting) than it had seemed at first sight

A cautionary tale ...(1)

- Chebshev mixing method (proposed by Erber, Everett and Johnson, *J. Comput. Phys.*, vol 32, p168 -, 1979)

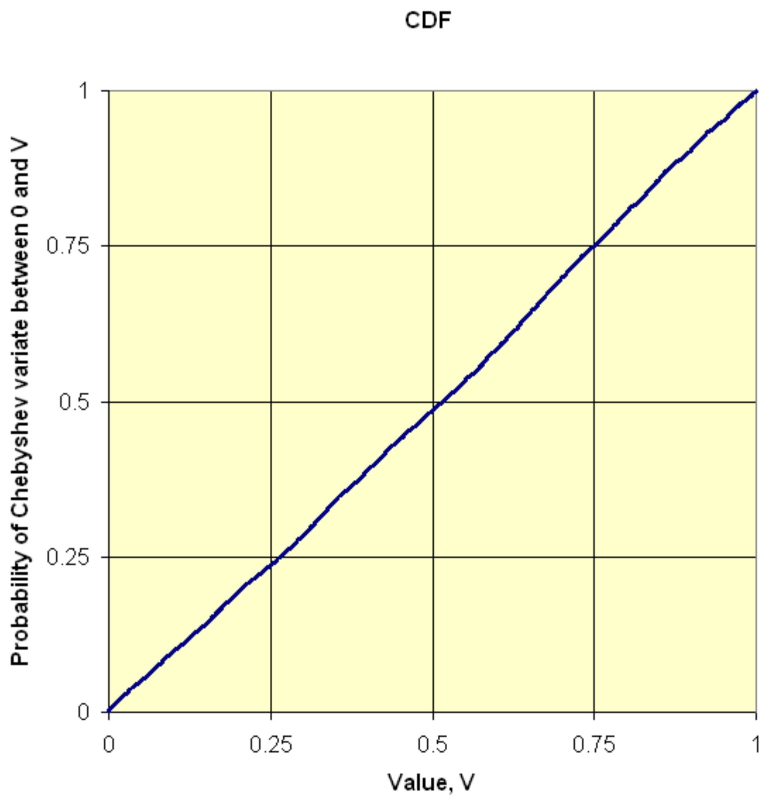
$$Z_n = Z_{n-1}{}^2 - 2 \text{ where initial } Z_0 \text{ lies in the range } (0,2)$$

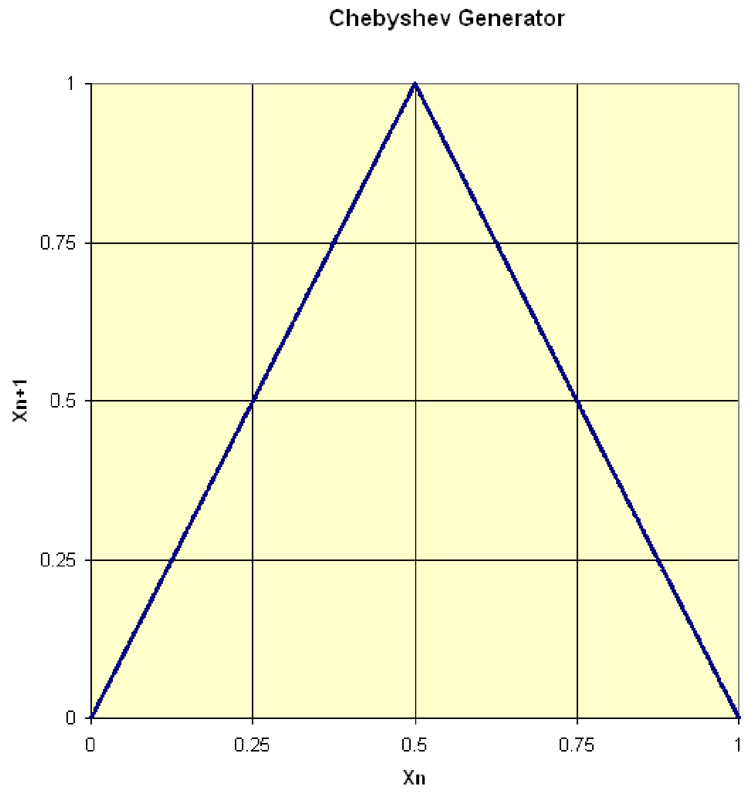
$$U_n = (1/\pi)\cos^{-1}(Z_n/2)$$

- Superficially, appears a good source of U(0,1) numbers
  - Simple, easy to implement
- BUT turns out to have undesirable qualities, making it unsuitable as a source of random numbers
  - As later pointed out by Erber et al, *J. Comput. Phys.*, 1983

RPS Energy

Chebyshev generator – distribution in one and two dimensions





Analysis of Chebyshev algorithm

- Can rewrite Chebyshev generator as

$$U_n = (1/\pi)\cos^{-1}(Z_n/2) = (1/\pi)\cos^{-1}\left((Z_{n-1}{}^2 - 2)/2\right)$$

$$\cos(\pi U_n) = (Z_{n-1}{}^2 - 2)/2 = 2(Z_{n-1}/2)^2 - 1$$

$$= 2\cos^2(\pi U_{n-1}) - 1 = \cos(2\pi U_{n-1})$$

- Hence, simplifies to

$$U_n = 2U_{n-1} \qquad U_{n-1} < 0.5$$

$$U_n = 2 - 2U_{n-1} \qquad U_{n-1} \geq 0.5$$

- Using exact finite precision arithmetic, with *k* binary digits, sequence collapses to zero after *k* steps
  - Only reason the generator 'works' at all is due to rounding error in inverse cosine calculation

A cautionary tale …(2)

- Linear congruential generator, LCG (see discussion in Knuth, The Art of Computer Programming, Vol 2. Semi-numerical algorithms)

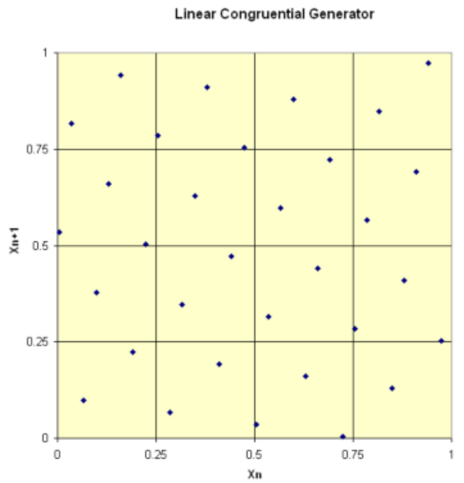
$$Y_n = \left(aY_{n-1} + c\right)_{\mathrm{mod}\,M} \qquad X_n = Y_n / M$$

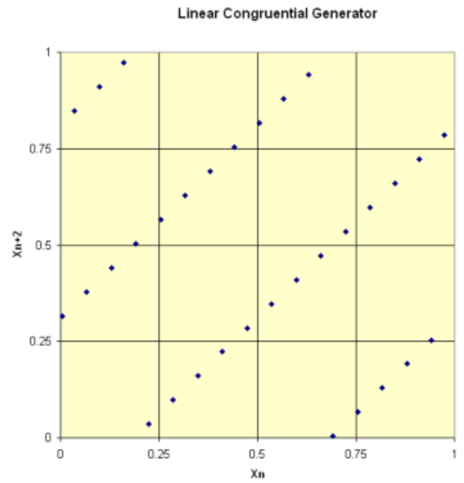
- Depends on appropriate choice of multiplier *a*, additive constant *c* and modulus *M*
  - For any given *M* only a very small proportion of choices of multiplier give good distribution properties
  - Extensive empirical testing required for each choice of *M*
  - Often restrict to generators with constant $c = 0$ (multiplicative congruential generator, MCG)
  - Period length always ≤ *M*

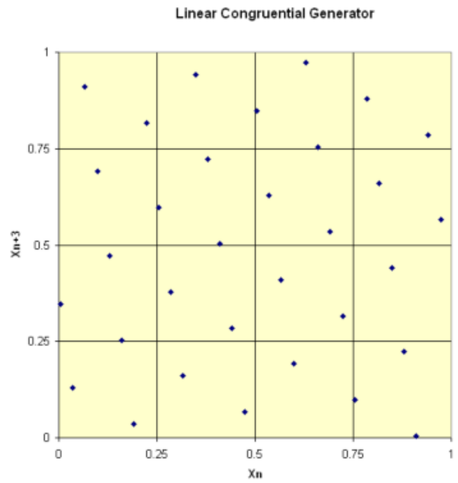
MCG issues (also apply to LCG)

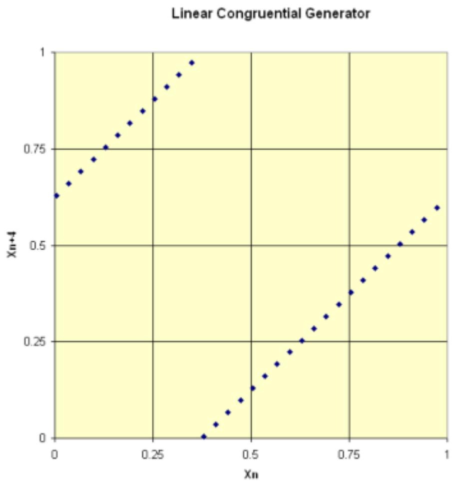
- With large *M* can get reasonable distribution properties in moderate number of dimensions and long period (but can only use a small fraction of full period)
  - Example: NAG routine G05CAF (modulus 2^{59}, multiplier 13^{13}; period length 2^{57}, provided seed is odd)
- To increase period, require increased modulus plus extensive empirical testing of large numbers of multipliers
  - No *a priori* way of predicting good multipliers
- For parallel processing, need much longer sequences (very large modulus) or many different statistically independent generators
- With smaller *M*, serious inadequacies with distribution properties
  - Many historical examples (smaller modulus) that were widely used and later turned out to have disastrous flaws on certain problems
    - eg RANDU (modulus 2^{31}, multiplier 65539, widely used in the scientific computing world for many years) but has very poor 3-d distribution
    - Might also have unforseen problems with current generators
  - Some examples follow

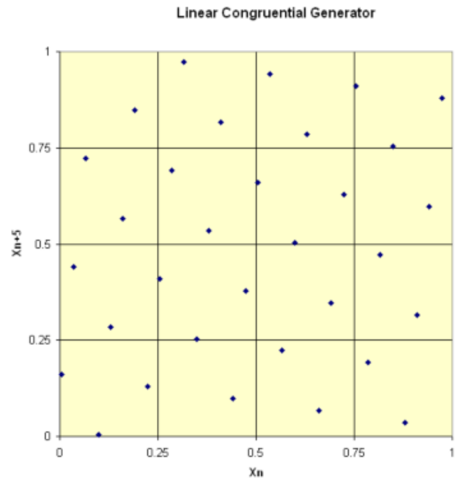
MCG, modulus 2^8=256, multiplier=137, initial value=1 (period=32)

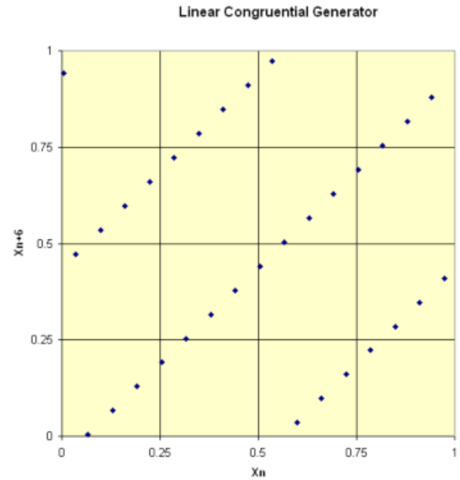




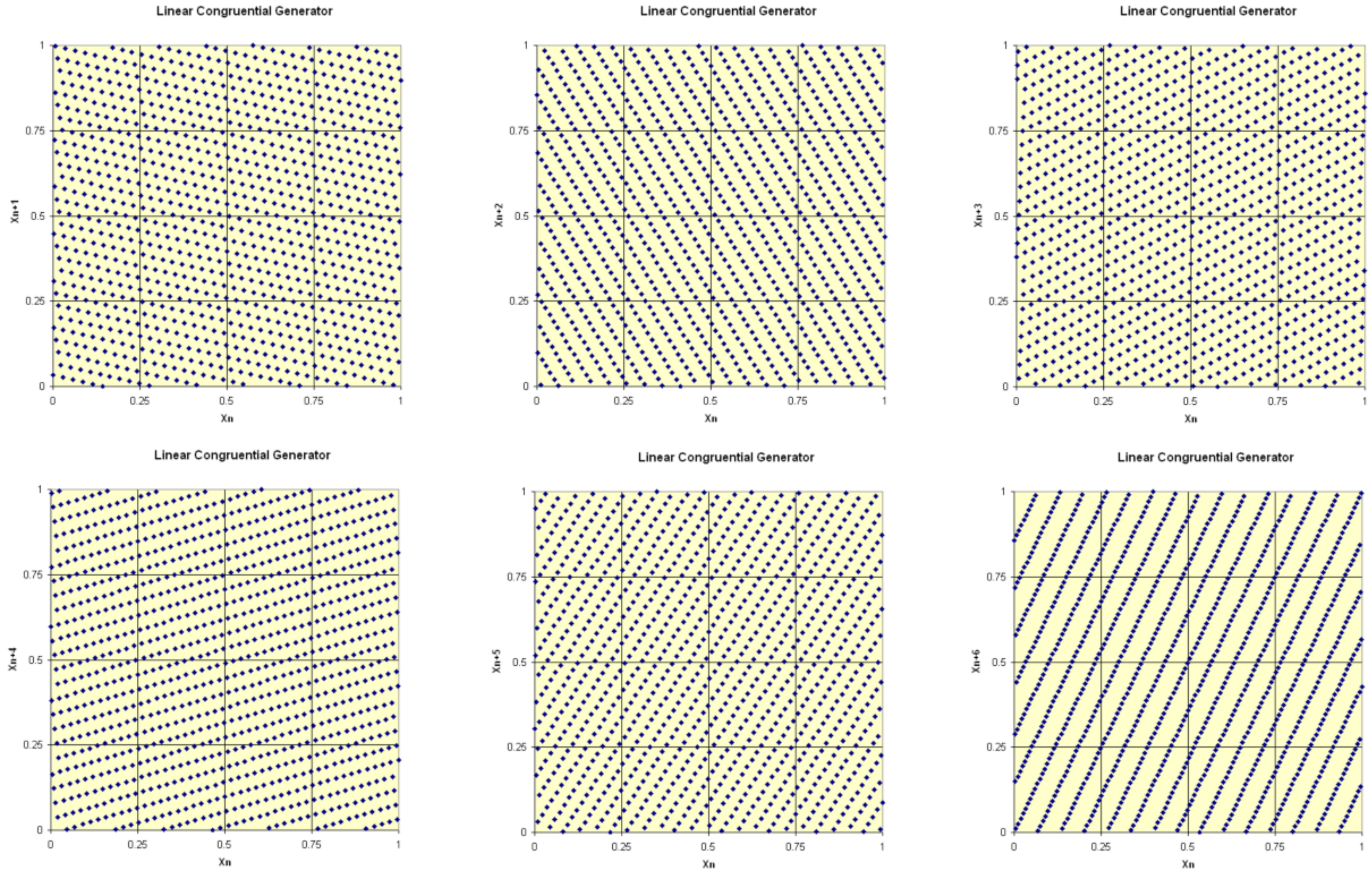








MCG, modulus 2^{12}=4096, multiplier=141, i.v. =1 (period=1024)



Additive congruential random number (ACORN) generator

- ACORN generator
  - Original discovery dates back to 1984/85
- Reference Wikramaratna, *J. Comput. Phys.*, vol 83, p16-31 (1989) and follow up papers
  - Simple to implement
  - Long period (≥ M; multiple of the modulus)
  - Amenable to theoretical analysis
  - k-th order generator approximates to k-distributed
    - in the sense that it can approximate arbitrarily closely to any specified finite number of terms from a sequence that can be proved to be k-distributed

ACORN random number generator

- *k*-th order ACORN generator defined from
  - an integer modulus M
  - an integer seed Y^0_0, $(0 < Y^0_0 < M)$
  - an arbitrary set of k integer initial values Y^m_0, $m = 1, ..., k$, each satisfying $0 \le Y^m_0 < M$

$$Y^0_n = Y^0_{n-1} \qquad n \ge 1$$

$$Y^m_n = (Y^{m-1}_n + Y^m_{n-1})_{\bmod M} \qquad n \ge 1, m = 1, \ldots, k$$

$$X^k_n = Y^k_n / M \qquad n \ge 1$$

RPS Energy

Calculating ACORN variates

$$
\begin{array}{lccccccccccccc}
Order\ 0: & Y^0{}_0 & \rightarrow & Y^0{}_1 & \rightarrow & Y^0{}_2 & \rightarrow & Y^0{}_3 & \rightarrow & Y^0{}_4 & \rightarrow & \ldots & \rightarrow & Y^0{}_n \\
 & & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & & & \downarrow \\
Order\ 1: & Y^1{}_0 & \rightarrow & Y^1{}_1 & \rightarrow & Y^1{}_2 & \rightarrow & Y^1{}_3 & \rightarrow & Y^1{}_4 & \rightarrow & \ldots & \rightarrow & Y^1{}_n \\
 & & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & & & \downarrow \\
Order\ 2: & Y^2{}_0 & \rightarrow & Y^2{}_1 & \rightarrow & Y^2{}_2 & \rightarrow & Y^2{}_3 & \rightarrow & Y^2{}_4 & \rightarrow & \ldots & \rightarrow & Y^2{}_n \\
 & & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & & & \downarrow \\
 & & & \ldots & & \ldots & & \ldots & & \ldots & & & & \ldots \\
 & & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & & & \downarrow \\
Order\ k: & Y^k{}_0 & \rightarrow & Y^k{}_1 & \rightarrow & Y^k{}_2 & \rightarrow & Y^k{}_3 & \rightarrow & Y^k{}_4 & \rightarrow & \ldots & \rightarrow & Y^k{}_n \\
 & & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & & & \downarrow \\
U[0,1): & & & X^k{}_1 & \rightarrow & X^k{}_2 & \rightarrow & X^k{}_3 & \rightarrow & X^k{}_4 & \rightarrow & \ldots & \rightarrow & X^k{}_n
\end{array}
$$

Some observations

- Numbers X^{k}_{n} approximate to uniformly distributed on the unit interval in up to k dimensions
  - provided a few simple constraints on initial parameter values are satisfied,
    - C1. Modulus M should to be a large integer (typically a prime number raised to an integer power)
    - C2. Seed Y^{0}_{0} and modulus should be relatively prime
    - C3. Initial values Y^{m}_{0} can then be chosen arbitrarily
  - Conditions C1 and C2 ensure a large period length (an integer multiple of the modulus).
- Suitable parameter combinations include
  - M a large prime; Y^{0}_{0} any integer smaller than M
  - $M = Q^{r}$ for prime Q & some integer r ; Y^{0}_{0} integer, not a multiple of Q
  - $M = 2^{30p}$ for some (small) integer p ; Y^{0}_{0} an odd integer
    - this last choice is particularly convenient, both for efficient implementation and theoretical analysis

Example implementations in FORTRAN 77, modulus 2^{30} or 2^{60}

```
      DOUBLE PRECISION FUNCTION ACORNI(XDUM)
C
C          ACORN GENERATOR
C          MODULUS =< 2^30, ORDER =< 12
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (MAXORD=12,MAXOP1=MAXORD+1)
      COMMON /IACO/ KORDEI,MAXINT,IXV(MAXOP1)
      DO 7 I=1,KORDEI
        IXV(I+1)=(IXV(I+1)+IXV(I))
        IF (IXV(I+1).GE.MAXINT)
     1        IXV(I+1)=IXV(I+1)-MAXINT
    7 CONTINUE
      ACORNI=(DBLE(IXV(KORDEI+1))/MAXINT
      RETURN
      END
```

- XDUM - dummy variable
- Common block IACO used to transfer data to the function
- Before first call, initialise variables in common block IACO (user must not subsequently change any of these parameters)
  - KORDEI - Order ≤ 12 (higher orders possible by increasing parameter MAXORD)
  - MAXINT - modulus for generator (≤ 2^{30}, to avoid integer overflow
  - IXV(1) - seed for generator (seed non-zero and < MAXINT, relatively prime with MAXINT; if MAXINT = 2^{30}, then IXV(1) must be odd)
  - IXV(I+1), I=2,KORDEI - initial values for generator (initial values< MAXINT)
- After initialisation, each call generates a single number between 0 and 1, returning it as the function value ACORNI.

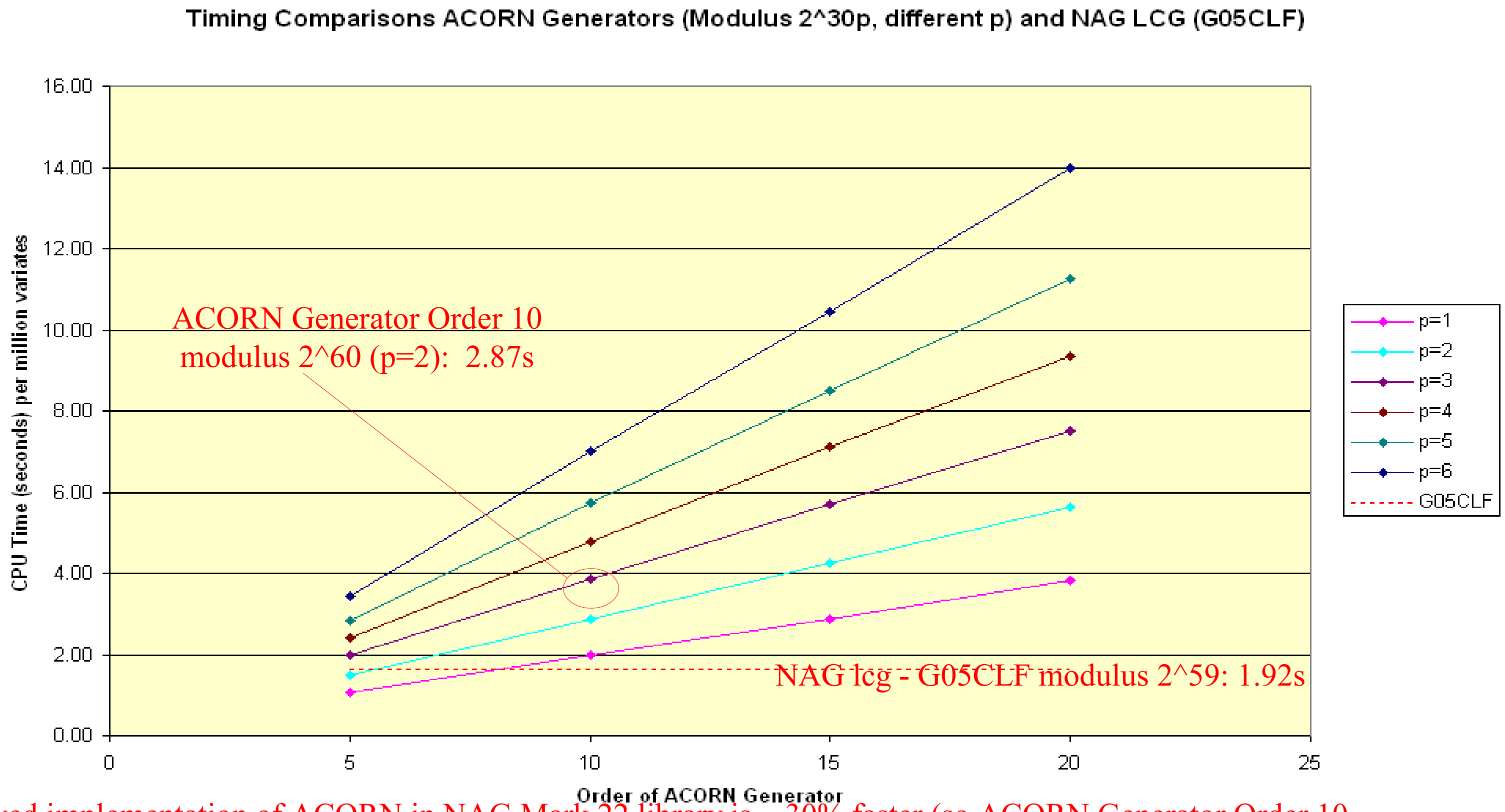
```
      DOUBLE PRECISION FUNCTION ACORNJ(XDUM)
C
C          ACORN GENERATOR
C          MODULUS =< 2^60, ORDER =< 12
C
      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
      PARAMETER (MAXORD=12,MAXOP1=MAXORD+1)
      COMMON /IACO2/ KORDEJ
     1    ,MAXJNT,IXV1(MAXOP1),IXV2(MAXOP1)
      DO 7 I=1,KORDEJ
        IXV1(I+1)=(IXV1(I+1)+IXV1(I))
        IXV2(I+1)=(IXV2(I+1)+IXV2(I))
        IF (IXV2(I+1).GE.MAXJNT) THEN
          IXV2(I+1)=IXV2(I+1)-MAXJNT
          IXV1(I+1)=IXV1(I+1)+1
        ENDIF
      IF (IXV1(I+1).GE.MAXJNT)
     1        IXV1(I+1)=IXV1(I+1)-MAXJNT
    7 CONTINUE
      ACORNJ=(DBLE(IXV1(KORDEJ+1))
     1   +DBLE(IXV2(KORDEJ+1))/MAXJNT)/MAXJNT
      RETURN
      END
```

NOTE – This is simplest and most easily understood implementation; significantly faster implementations are possible (eg NAG Fortran Library, Mark 22) while still producing identical sequences for any specified implementation

Computational performance (Martyn Byng, NAG, circa 2007)

Time to exhaust period with single processor:
ACORN modulus 2^30 ~ 0.1 to 0.8 days
ACORN modulus 2^60 ~ 0.3 to 3.5 million years

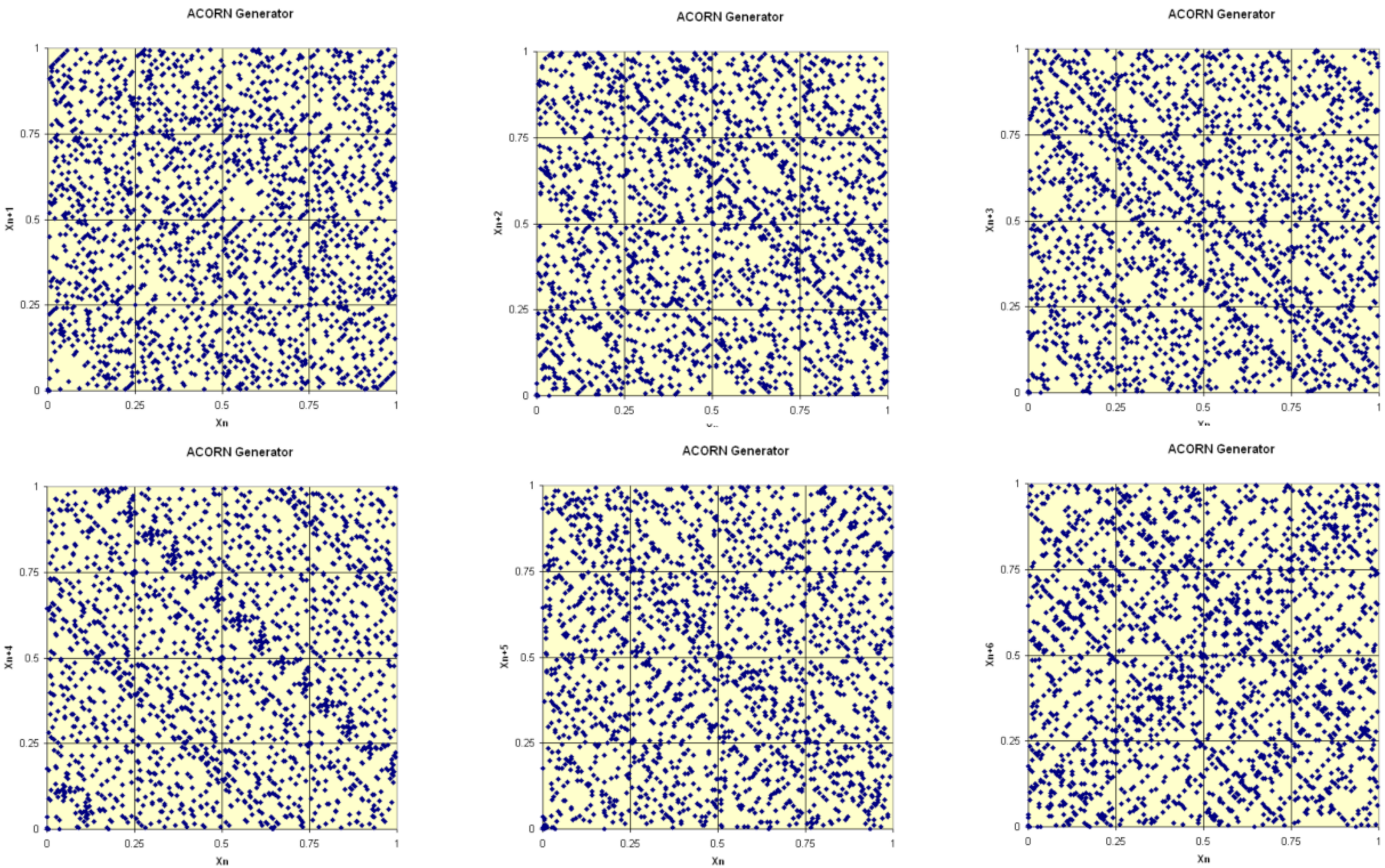
(Timings on:Windows 2000 Professional on Pentium III 600MHz processor with 128Mb memory using Compaq Visual Fortran 6 Compiler)



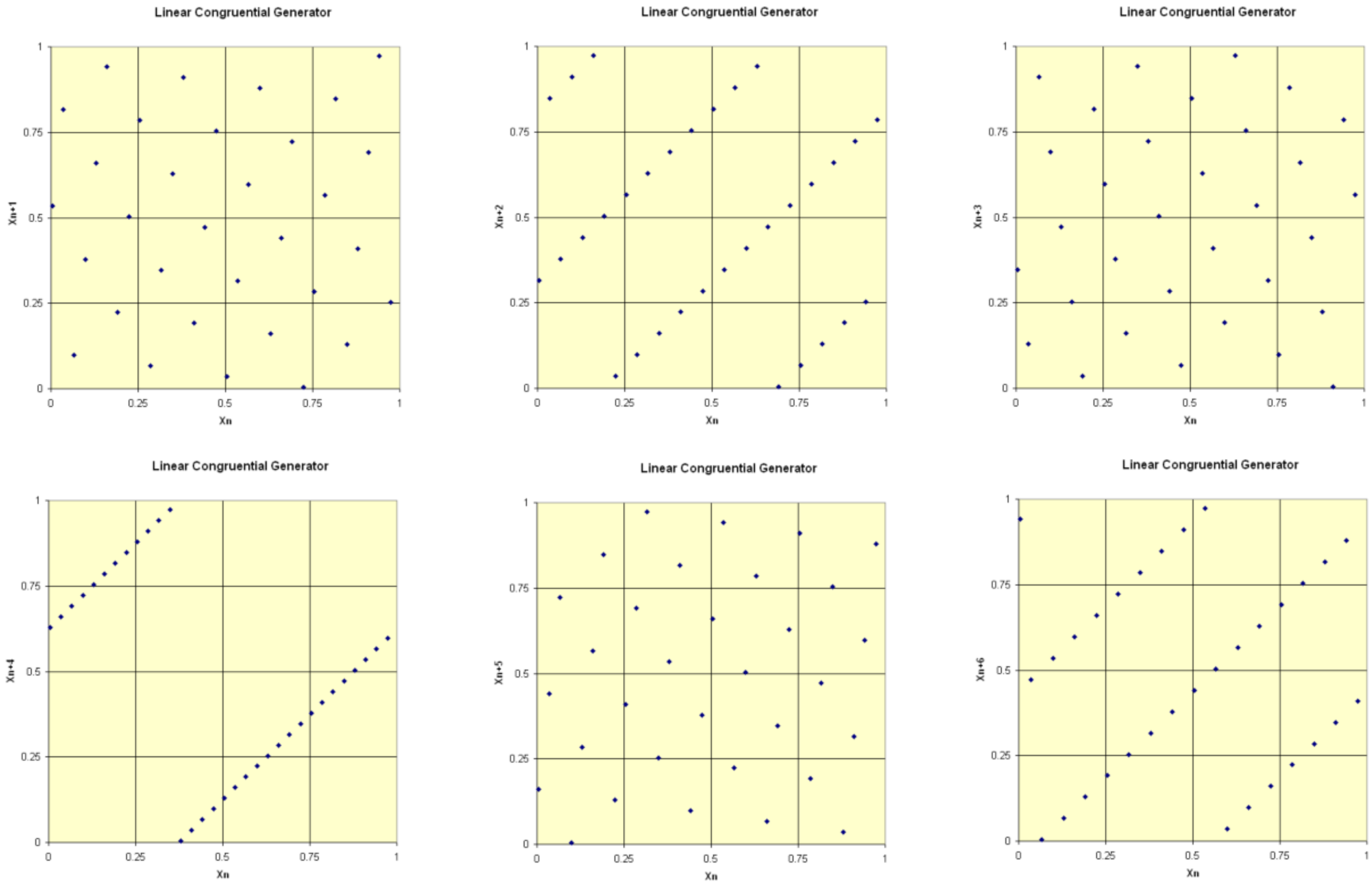
NOTE: improved implementation of ACORN in NAG Mark 22 library is ~ 30% faster (so ACORN Generator Order 10 modulus 2^60 (p=2) is comparable with G05CLF); Mersenne Twister implementation also gives comparable performance. Significant further speedup possible for all approaches by generating more than one variate per call.

RPS Energy

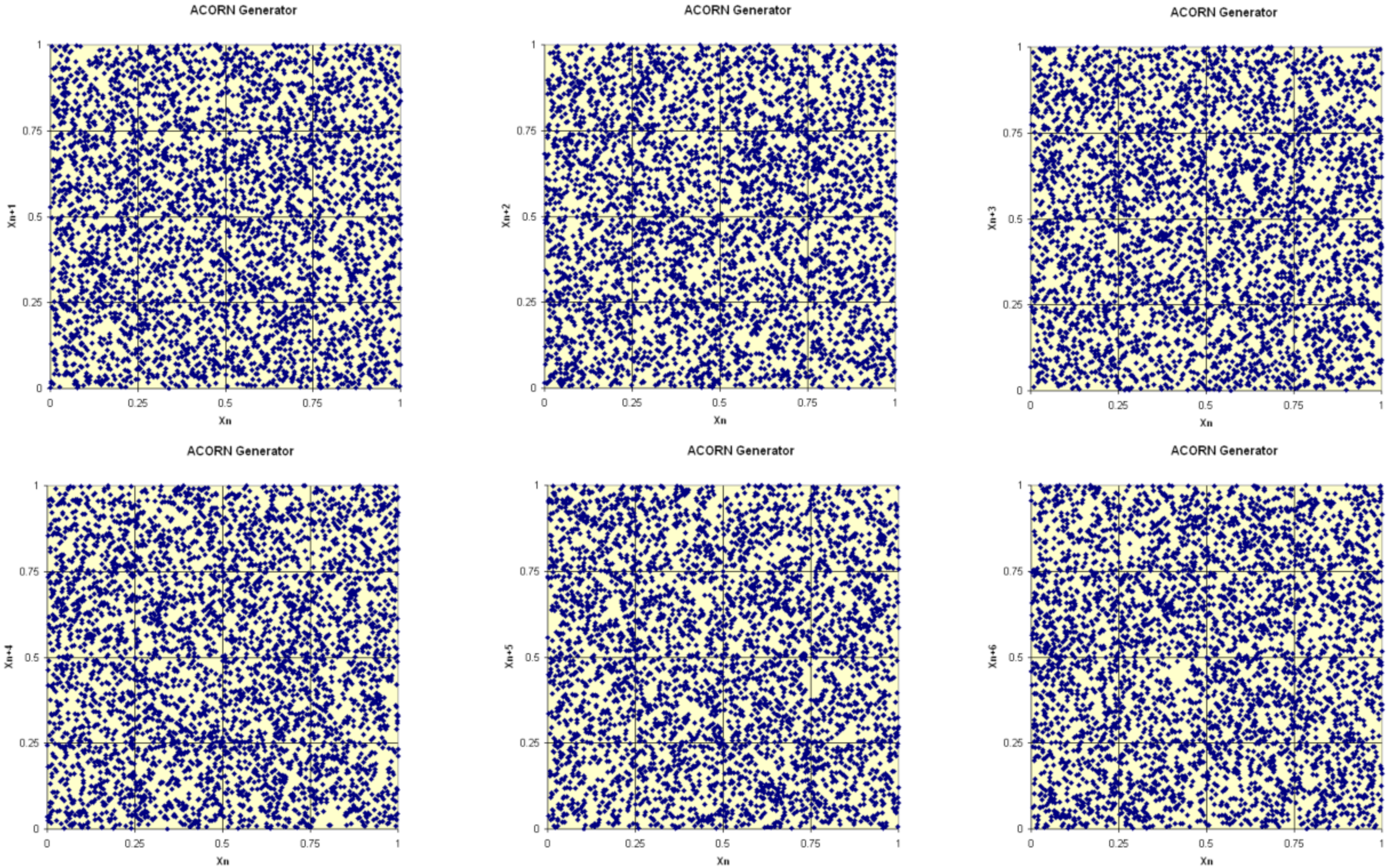
ACORN, modulus 2^8=256, order 8 (period=8x256=2024)



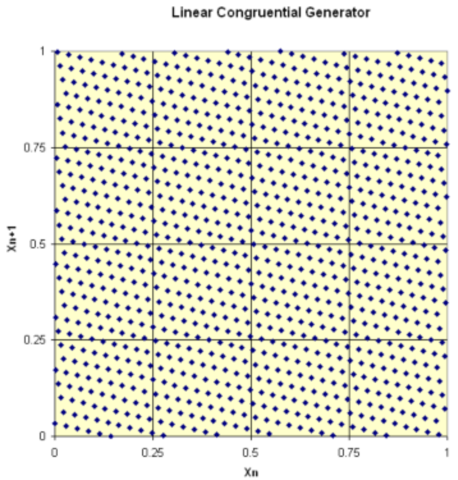
MCG, modulus 2^8=256, multiplier=137, initial value=1 (period=32)

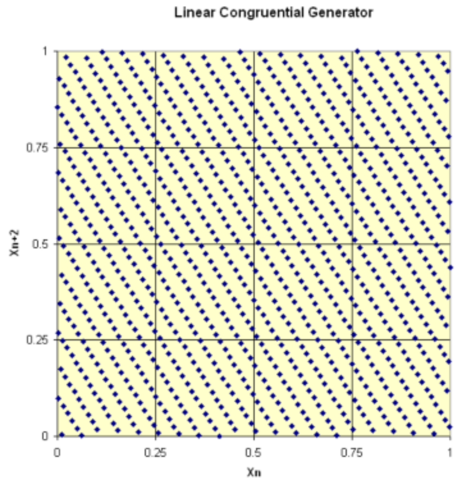


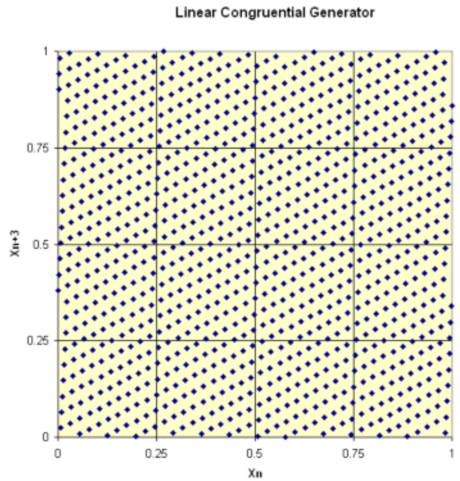
ACORN, modulus 2^{12}=4096, order 10 (period=8x4096; first 4096 points only)

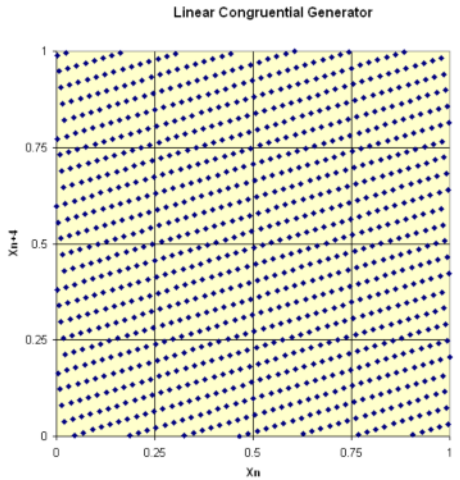


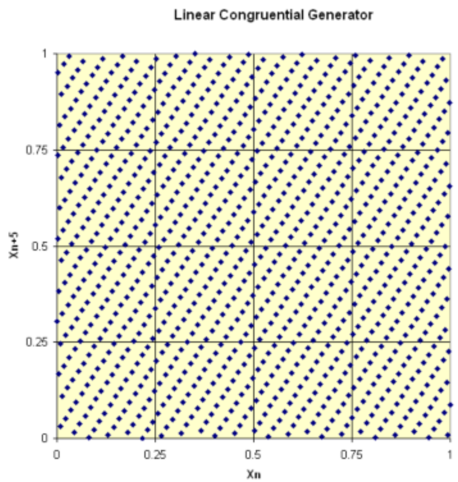
MCG, modulus 2^{12}=4096, multiplier=141, i.v. =1 (period=1024)

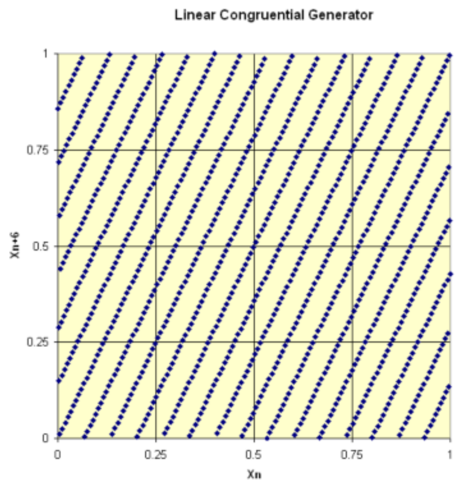












Empirical testing (modulus ≥ 2^{60}, order ≥ 10)

- Computational Physics example, ~2000
  - Simulation of 2D Ising model, using cluster algorithms and in particular the Wolff algorithm [U. Wolff, Phys. Rev. Lett., 62, 361, 1989].
  - A.M. Ferrenberg, D.P. Landau and Y.J. Wong [Phys. Rev. Lett., 69, 3382, 1992] demonstrated that a number of supposedly 'high quality' random number generators produced systematically incorrect results on this problem.
  - M. Luscher [Computer Physics Communications, 79, 100, 1994] has suggested that this is a particularly sensitive test of random number generators.
  - Tests reported by Ferrenberg et al and by Luscher were repeated using ACORN algorithm as source of random numbers [U. Wolff, personal communication, 2000]. Discrepancy between simulation results and the exact analytic solution was statistically insignificant - ACORN generator passed this test (good LC generators also pass test)
- ~ 2005: using Diehard (Marsaglia, 1995)
  - See Wikramaratna, 2008a, submitted to JCAM
- ~ 2008: using TestU01 (L'Ecuyer and Simard, 2007)
  - See Wikramaratna, 2008b, submitted to JCAM

Theoretical analysis and results

- Main theoretical results to date
  - Closed form expression for *n*-th term
  - Periodicity (note larger than LCG with similar modulus)
  - Parallelisation of Monte-Carlo calculations
  - Equivalence between ACORN and certain specific multiple recursive and matrix generators
  - *k*-th order ACORN generator approximates to *k*-distributed

Closed-form expression for *n*-th term in ACORN sequence

Define $Z^m{}_n$ as follows

$$Z^0{}_n = 1 \qquad Z^1{}_n = n \qquad Z^2{}_n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$Z^3{}_n = \sum_{i=1}^{n} \frac{i(i+1)}{2} = \frac{n(n+1)(n+2)}{3!} \qquad \ldots \qquad Z^m{}_n = \frac{(n+m-1)!}{(n-1)!m!}$$

Leads to the following closed form expression for $Y^m{}_n$

$$Y^m{}_n = \left(\sum_{i=0}^{m} Y^i{}_0 Z^{m-i}{}_n \right)_{\mathrm{mod}\, M} \qquad \text{where} \qquad Z^{m-i}{}_n = \frac{(n+m-i-1)!}{(n-1)!(m-i)!}$$

Period length (1989)

- Have proved that the period length of an ACORN sequence with modulus equal to a power of two will be an integer multiple of the modulus, provided only that the seed is chosen to be odd.
  - Period length of the sequence can be increased, effectively without limit, simply by increasing the value of the modulus by a suitable factor and then choosing the seed to take an odd value.
  - Implementation is straightforward, for arbitrarily large modulus
- Contrast with MCG/LCG for which the period length can never exceed the modulus
  - Increasing the modulus for a linear congruential generator is non-trivial as a result of need to identify appropriate new values of the parameters *a* and *c* in order to ensure reasonable distribution properties in higher dimensions
  - implementation of a linear congruential generator becomes progressively more complicated with increasing modulus.

Conjecture (2008) on period length

- Let X^k_n be a k -th order ACORN generator, with modulus equal to a prime power ($M = q^{\,t}$, where q is a prime) and suppose the seed and modulus are relatively prime. Then the sequence X^k_n, $k = 1, \ldots, n$ will have period length equal to $q^{\,i}M = q^{\,i+t}$, where i is the largest integer such that $q^{\,i} \leq k$.
- Observations
  - Seed and modulus relatively prime means seed should not be a multiple of q
  - If $q = 2$, then condition satisfied provided only that q takes odd value
  - Result holds irrespective of initial values
  - Restriction on seed is necessary (consider case with seed equal to q and all initial values zero – first order generator has period M/q
- Examples (see also next slide)
  - Modulus 2^{60}, order 10 gives period 2^{63}
  - Modulus 2^{90}, order 16 gives period 2^{94}

Period lengths for various moduli

$M = 2^t$		$M = 3^t$		…	$M = q^t$, q prime	
Order k	Period	Order k	Period		Order k	Period
$k=1$	M	$1 \le k < 3$	M		$1 \le k < q$	M
$2 \le k < 4$	$2M$	$3 \le k < 9$	$3M$		$q \le k < q^2$	qM
$4 \le k < 8$	$4M$	$9 \le k < 27$	$9M$		$q^2 \le k < q^3$	$q^2 M$
…		…		…	…	
$2^i \le k < 2^{i+1}$	$2^i M$	$3^i \le k < 3^{i+1}$	$3^i M$		$q^i \le k < q^{i+1}$	$q^i M$

RPS Energy

Parallelisation of Monte-Carlo calculations (2000)

- Approaches to parallelisation
  - Parameterisation (define family of random number generators having a parameter that can be varied between different processors)
    - Success dependant on statistical independence of the different generators
  - Splitting (output from a single random number generator with long period is split into a number of sub-streams which can then be used either on different processors or for different realisations of the Monte-Carlo calculation)
    - If number of variates per realisation is known (or has a known bound) then can do identical calculation on any number *p* of processors with speed-up factor very close to *p*
    - Needs efficient algorithm to take large strides (fixed length) through the sequence of random numbers
      - ‘Efficient’ means much faster than stepping through the sequence term by term
      - Have demonstrated how to do this for ACORN
    - Need very long period length (sufficient to carry out full set of realisations)
      - Can always choose ACORN generator with sufficient period length

Equivalence with multiple recursive generators (2008) …

- Generalised MRG, order *k*: each variate is a linear combination of previous *k* variates and a constant
  - Normalise to the unit interval by dividing by *M*

$$y_j = \left(a_1 y_{j-1} + a_2 y_{j-2} + \ldots + a_k y_{j-k} + c\right)_{\mathrm{mod}\,M} \qquad x_j = y_j / M$$

- *k*-th order ACORN is equivalent to a *k*-th order generalised MRG with coefficients a^i_j and *c* where
  - a^i_j alternate in sign; magnitude first increases, then decreases

$$a_i = \left((-1)^{i+1}\frac{k!}{(k-i)!i!}\right)_{\mathrm{mod}\,M} \qquad i = 1,\ldots,k; \qquad c = Y^0{}_0$$

RPS Energy

… and with matrix generators (also 2008)

- Generalised MRG can be re-written

$$\begin{pmatrix} y_{j-k+1} \\ y_{j-k+2} \\ \vdots \\ y_{j-1} \\ y_j \end{pmatrix} = \left[\begin{pmatrix} 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & \ddots & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & \ldots & 0 & 1 \\ a_k & a_{k-1} & \ldots & a_2 & a_1 \end{pmatrix} \begin{pmatrix} y_{j-k} \\ y_{j-k+1} \\ \vdots \\ y_{j-2} \\ y_{j-1} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ c \end{pmatrix} \right]_{\mathrm{mod}\, M}$$

$$\mathbf{y}_j = \left(\mathbf{G}_k \mathbf{y}_{j-1} + \mathbf{c}\right)_{\mathrm{mod}\, M}$$

$$\mathbf{y}_j = \left([\mathbf{G}_k]^k \mathbf{y}_{j-k} + \left([\mathbf{G}_k]^{k-1} + \ldots + [\mathbf{G}_k]^1 + [\mathbf{I}_k]\right)\mathbf{c}\right)_{\mathrm{mod}\, M}$$

$$= \left([\mathbf{G}_k]^k \mathbf{y}_{j-k} + [\mathbf{B}_k]\mathbf{c}\right)_{\mathrm{mod}\, M} = \left([\mathbf{G}_k]^k \mathbf{y}_{j-k} + \mathbf{b}_k c\right)_{\mathrm{mod}\, M}$$

  - Apply *k* times – gives matrix equation with <u>disjoint</u> vectors
  - This is a particular case of a <u>matrix generator</u> (with matrix $[\mathbf{G}_k]^k$)
    - Can study the form of the matrices ($\mathbf{G}_k$, $[\mathbf{G}_k]^k$, $\mathbf{B}_k$), vector $\mathbf{b}_k$ and the magnitude of the coefficients (see paper)
    - Observe in particular that $\mathbf{G}_k^{-1} = \mathbf{G}_k^R$ where ^{R} denotes reversing order of rows <u>and</u> columns (equivalently, rotating by 180° about mid-point of matrix)

k-distributed property (1992)

$$\text{Normalised form of ACORN generator:}$$

$$X^0{}_n = X^0{}_{n-1} \qquad n \geq 1$$

$$X^m{}_n = (X^{m-1}{}_n + X^m{}_{n-1})_{\mathrm{mod}1} \qquad n \geq 1, m = 1,\ldots,k$$

- The *k*-th order ACORN random number generator (normalised to the unit interval) is well distributed modulo 1 in $\Re^k$, provided that the seed is irrational
  - Contrast with much weaker corresponding result for LCG – can show that a normalised LCG can at best be uniformly distributed (but NOT well distributed) modulo 1 in $\Re$, and is NOT uniformly distributed (or well distributed) modulo 1 in $\Re^k$ for any $k>1$
- Although any practical implementation uses rational seed, this suggests that with large enough modulus ACORN sequences can provide good approximation to *k*-distributed

RPS Energy

k-distributed convergence (2008)

- Given an arbitrary k -th order ACORN sequence together with modulus $M = 2^m$ and an appropriate set of initial conditions (including an odd value for the seed), together with a required precision $b \le m$; then the first M terms of the sequence are equal (to b binary digits precision) to the first M terms of an infinite sequence that is w.d. mod 1 in $\Re^k$.
- Given any normalised k -th order ACORN sequence together with an appropriate set of initial conditions (in particular, with an irrational seed χ^0_0 – which ensures that the sequence is is w.d. mod 1 in $\Re^k$), then we can calculate the first $N = 2^\nu$ terms of the sequence to β binary digits accuracy from an ACORN sequence with appropriate values of the modulus, seed and initial values.
- These results formalise the notion that "with large enough modulus ACORN sequences can provide good approximation to k-distributed"
- Hence conclude that a k-th order ACORN generator will give good results for Monte-Carlo integration in k dimensions, <u>for any k</u>.

Where Next?

- Tremendous scope for further theoretical analysis of ACORN algorithm
  - Needs a concentrated effort to see how far theory can be developed
  - Limits on how fast it can be developed by one person working occasionally (or even obsessively) in spare time
  - Great opportunity to look at some very interesting applications of mathematics, and to make a real impact
- Numerical Algorithms Group have included ACORN generator in latest release of NAG subroutine libraries (Mark 22, 2009)
  - Focus for more extensive testing and use on wider range of real applications in the future
  - Comparison with other leading algorithms (eg Mersenne Twister, see Matsumoto and Nishimura, 1998)

Some Research Opportunities

- Does ‘convergence’ property for ACORN generators give a real practical benefit?
- Proof of conjecture on periodicity
- Conditions for $\mathbf{G}_k^{-1} = \mathbf{G}_k^{R}$ where ^{R} denotes reversing order of rows and columns; other properties of such matrices; do such matrices occur elsewhere
- More efficient implementations (but note already dominated by overhead from subroutine calls – so can already get very significant benefits from a program design that allows multiple variates to be generated in one call)
- Practical applications – tests on real problems requiring random numbers; results comparison with other algorithms (in particular, the Mersenne Twister)

Conclusions

- ACORN algorithm appears to provide a practical source of *k*-distributed pseudo-random numbers for any *k*
  - Extremely simple to implement, in particular for modulus a power of 2
  - Period length a multiple of modulus (provided seed and modulus relatively prime) – can be increased without limit
  - Identical sequences on any machine (to available machine precision)
  - Splitting approach allows parallelisation of Monte-Carlo calculations
- ACORN algorithm gives rise to some very interesting mathematical analysis that demonstrates *a priori* that the sequences will have the desired properties
  - Reduces the need for extensive empirical testing
  - Allows test of results by repeating calculations with a different (higher order, larger modulus) ACORN sequence and comparing results

RPS Energy

ACORN References

- ACORN - A New Method for Generating Sequences of Uniformly Distributed Pseudo-random Numbers, *J. Comput. Phys.*, **83** (1989) 16-31
- Theoretical Background for the ACORN Random Number Generator, Report AEA-APS-0244, AEA Technology, Winfrith, Dorset, UK (1992)
- Pseudo-random Number Generation for Parallel Processing – A Splitting Approach, *SIAM News*, **33** number 9 (2000)
- The Additive Congruential Random Number Generator – a Special Case of a Multiple Recursive Generator, *J. Comput. and Appl. Mathematics*, **216** (2008) 371-387 (doi: 10.1016/j.cam.2007.05.018)
- Empirical Testing of the Additive Congruential Random Number Generator, unpublished (submitted to *J. Comput. and Appl. Mathematics*, January 2008)
- Theoretical and Empirical Convergence Results for Additive Congruential Random Number Generators, unpublished (submitted to *J. Comput. and Appl. Mathematics*, July 2008)