RPS Energy

Specification, Implementation and Numerical Analysis of a *k*-Distributed Uniform Pseudo-random Number Generator (ACORN)

Roy S Wikramaratna

WikramaratnaR@rpsgroup.com

Applied Mathematics and Numerical Analysis Seminar, Reading University

22 February 2008

RPS Energy

Acknowledgements

- Chris Farmer (Schlumberger and Oxford University)
  - Suggesting problem (1984)
  - Continuing interest in progress and results
- RPS Energy
  - Current employer
  - For giving me the time to present seminar
- 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

RPS Energy

Some initial thoughts

- Consider the equation $$Y^{m}{}_{n} = (Y^{m-1}{}_{n} + Y^{m}{}_{n-1})_{\mathrm{mod}\, M}$$
  - Some more information required – for example
    - Specify ranges for indices *m*, *n*
      - $m = 1, 2, \ldots, k$; $n = 1, 2, \ldots$
    - Specify initial values $Y^{0}{}_{n}$ and $Y^{m}{}_{o}$; any constraints on values taken?
    - *Y,M* are integers; any constraints on *M*? Or *Y* real, *M*=1?
- Is there any useful (and/or interesting) mathematics that comes out of studying this equation?
  - What exactly do we mean by useful?
    - Does it arise in a real problem?
    - Does studying it help us to solve any real problems?

RPS Energy

Outline

- Background
  - Pseudo-random number generation
  - 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 (gives uniformity in *k*-dimensions, any given *k*; period length in excess of any given number)
  - Gives rise to some very interesting analysis and useful mathematical results

RPS Energy

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 optimisation
    - numerical integration
    - 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

RPS Energy

Motivation

- 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

RPS Energy

A cautionary tale ...(1)

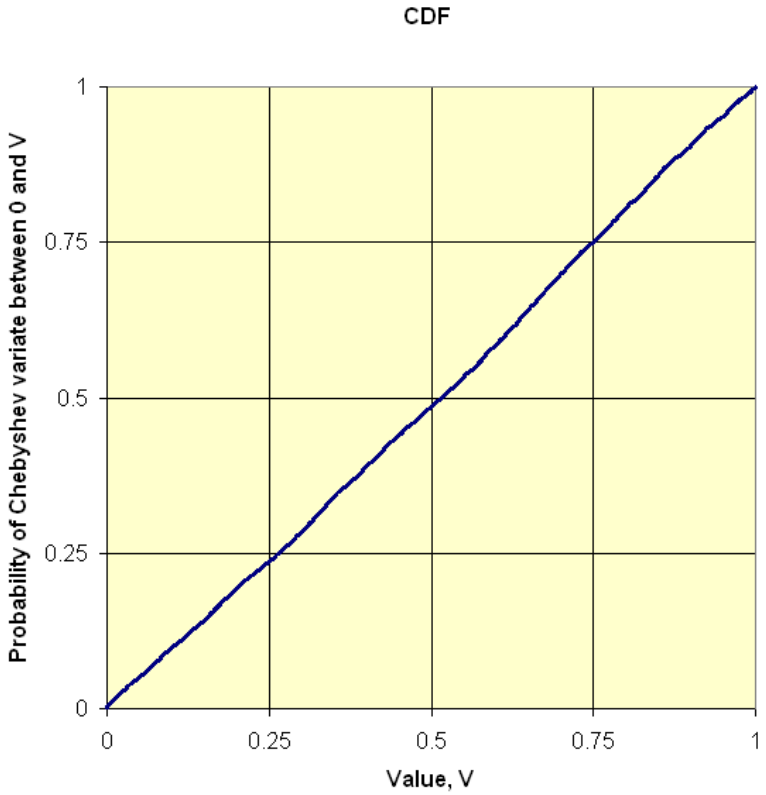
- Chebshev mixing method (proposed by Erber, Everett and Johnson, *J. Comput. Phys.*, vol 32, p168 -, 1979)

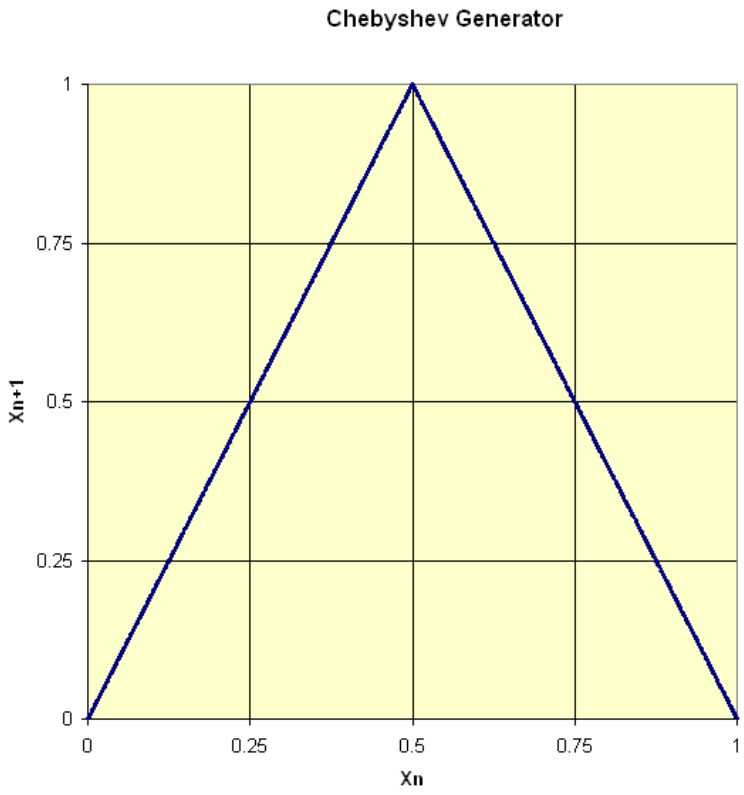
$$Z_n = Z_{n-1}{}^2 - 2 \text{ where initial } Z_0 \text{ lies in the range } (0,2)$$

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

- 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

Chebyshev generator – distribution in one and two dimensions





RPS Energy

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

RPS Energy

Observations

- Many different generators have been proposed over the years which initially appeared to pass a range of empirical tests of uniformity and randomness but which later turned out to have serious inadequacies in certain other specific tests of randomness
- It might seem these pitfalls could be largely overcome if it were possible to prove purely from theoretical considerations that a particular algorithm would pass certain classes of test, without the need for extensive empirical testing
  - It would be nice to have a family of generators, defined by some parameter, which 'converged' to uniform distribution (in *k* dimensions) as a limiting case for that parameter
  - Could then repeat calculations with different sequences (defined by different values of the relevant parameter) and check for convergence of the result
    - eg Monte-Carlo integration in *k*-dimensions

RPS Energy

A cautionary tale ...(2)

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

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

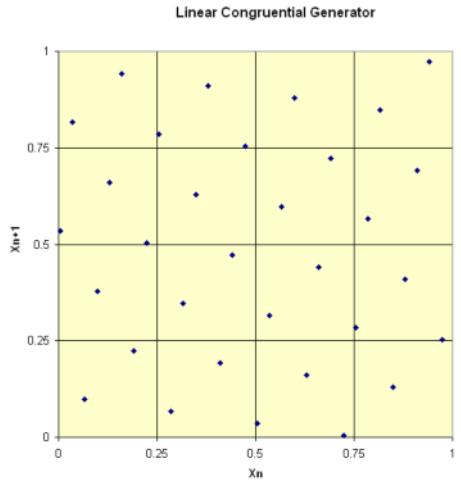
- Depends on appropriate choice of multiplier *a*, additive constant *c* and modulus *M*
  - For any given *M* only a very small proportion of choices of multiplier give good distribution properties
  - Extensive empirical testing required for each choice of *M*
  - Often restrict to generators with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

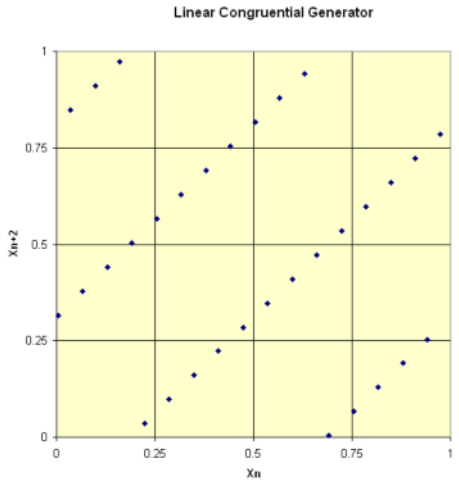
RPS Energy

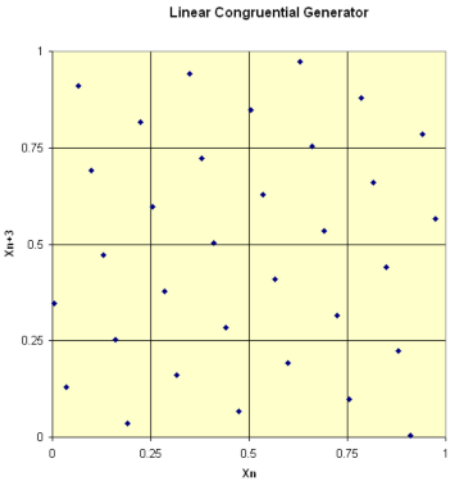
MCG issues (also apply to LCG)

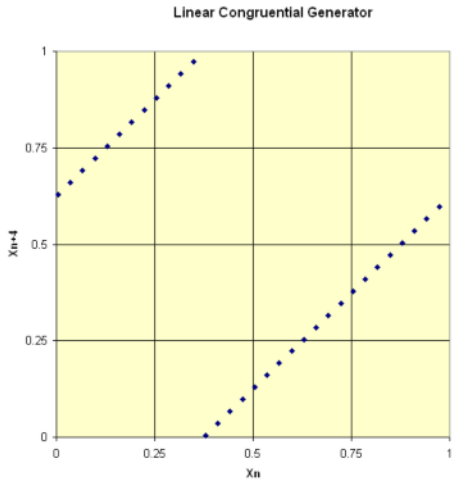
- With large *M* can get reasonable distribution properties in moderate number of dimensions and long period
  - Example: NAG routine G05CAF (modulus 2^{59}, multiplier 13^{13}; period length 2^{57}, provided seed is odd)
- To increase period, require increased modulus plus extensive empirical testing 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

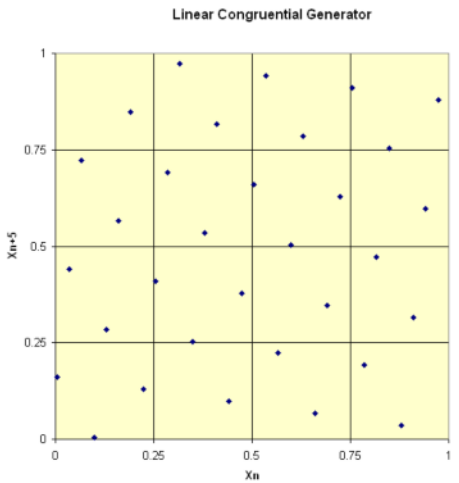
MCG, modulus 2^8=256, multiplier=137, initial value=1 (period=32)

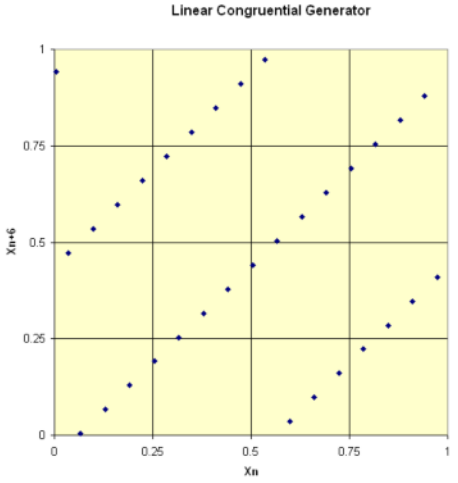




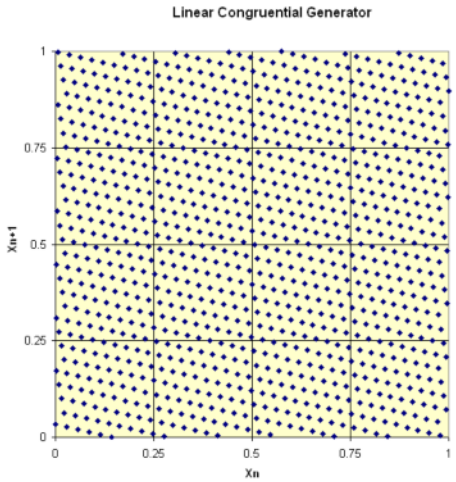


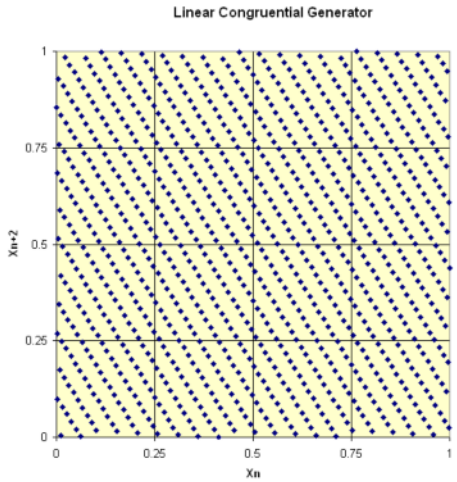


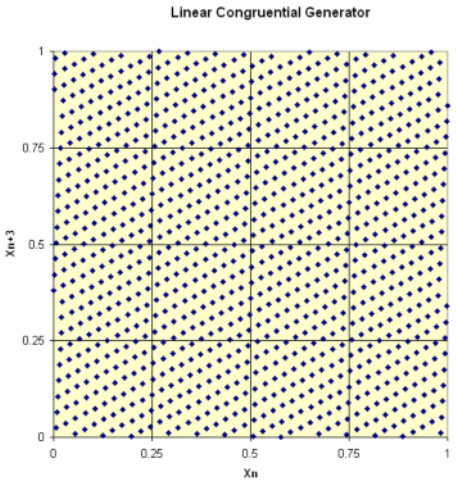


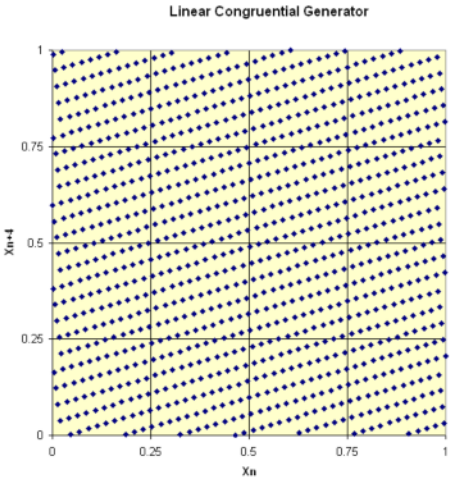


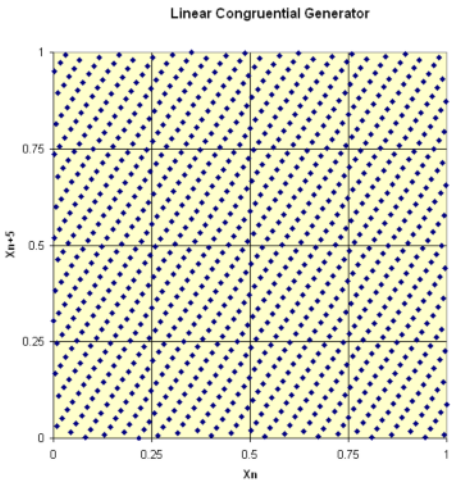
MCG, modulus 2^{12}=4096, multiplier=141, i.v. =1 (period=1024)

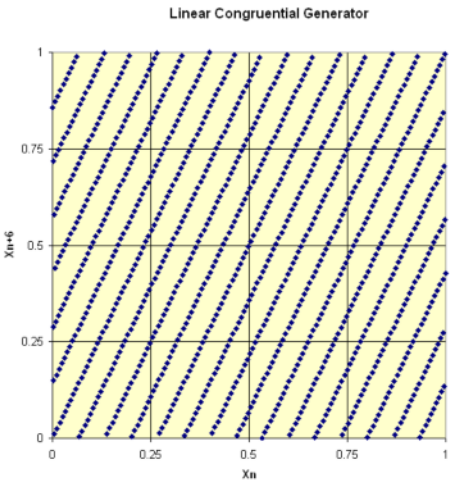












RPS Energy

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

RPS Energy

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, $m = 1, \ldots, k$, each satisfying $0 \le Y^m_0 < M$

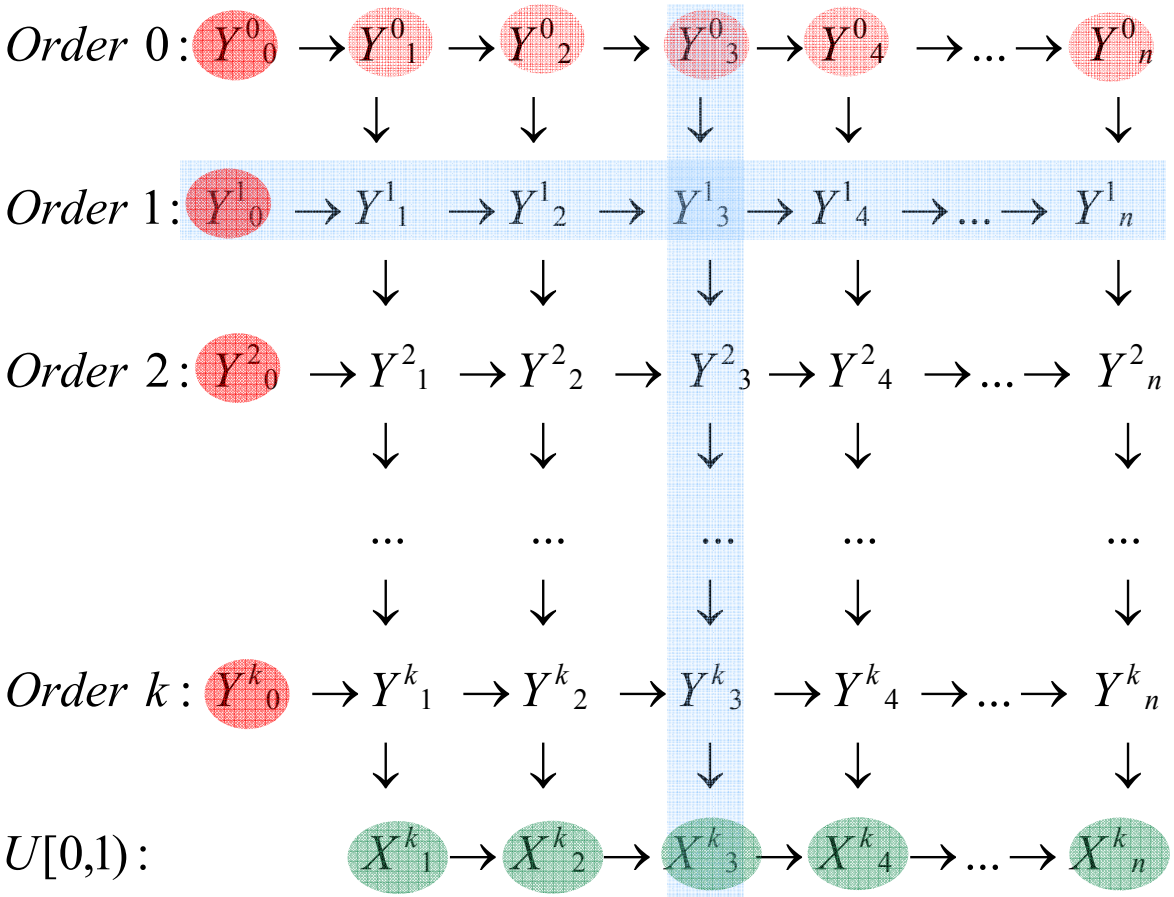
$$Y^0{}_n = Y^0{}_{n-1} \qquad n \ge 1$$

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

$$X^k{}_n = Y^k{}_n / M \qquad n \ge 1$$

RPS Energy

Calculating ACORN variates



RPS Energy

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).

RPS Energy

Suitable parameter choices

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

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

$$X^k{}_n = Y^k{}_n / M \qquad n \geq 1$$

- Suitable parameter combinations include
  - M a large prime; $Y^0{}_0$ any integer smaller than M
  - $M = Q^r$ for prime Q and some integer r ; $Y^0{}_0$ any 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

RPS Energy

Implementation of ACORN algorithm

- Simple to implement in any high-level language
  - assumes that integer representation allows integers up to 2^{31} to be calculated and stored without overflow
  - implement for modulus $M=2^{30p}$, integer p = 2, 3, or 4
- All computations performed in exact integer arithmetic, apart from conversion from integer modulo *M* to double precision real
  - identical results on any machine and/or language (to the accuracy of the machine representation of a double precision real number).
- Examples in FORTRAN 77 - implement as a function call with fewer than 20 lines of executable code
  - Analogous implementations in both C and C++
- NAG plan to introduce ACORN algorithm in next release of their subroutine libraries
  - Version currently available for download and testing at *http://www.nag.co.uk/nagware/Examples/Acorn.asp*

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
```

RPS Energy

Extension to larger modulus 2^{60}, 2^{30p}

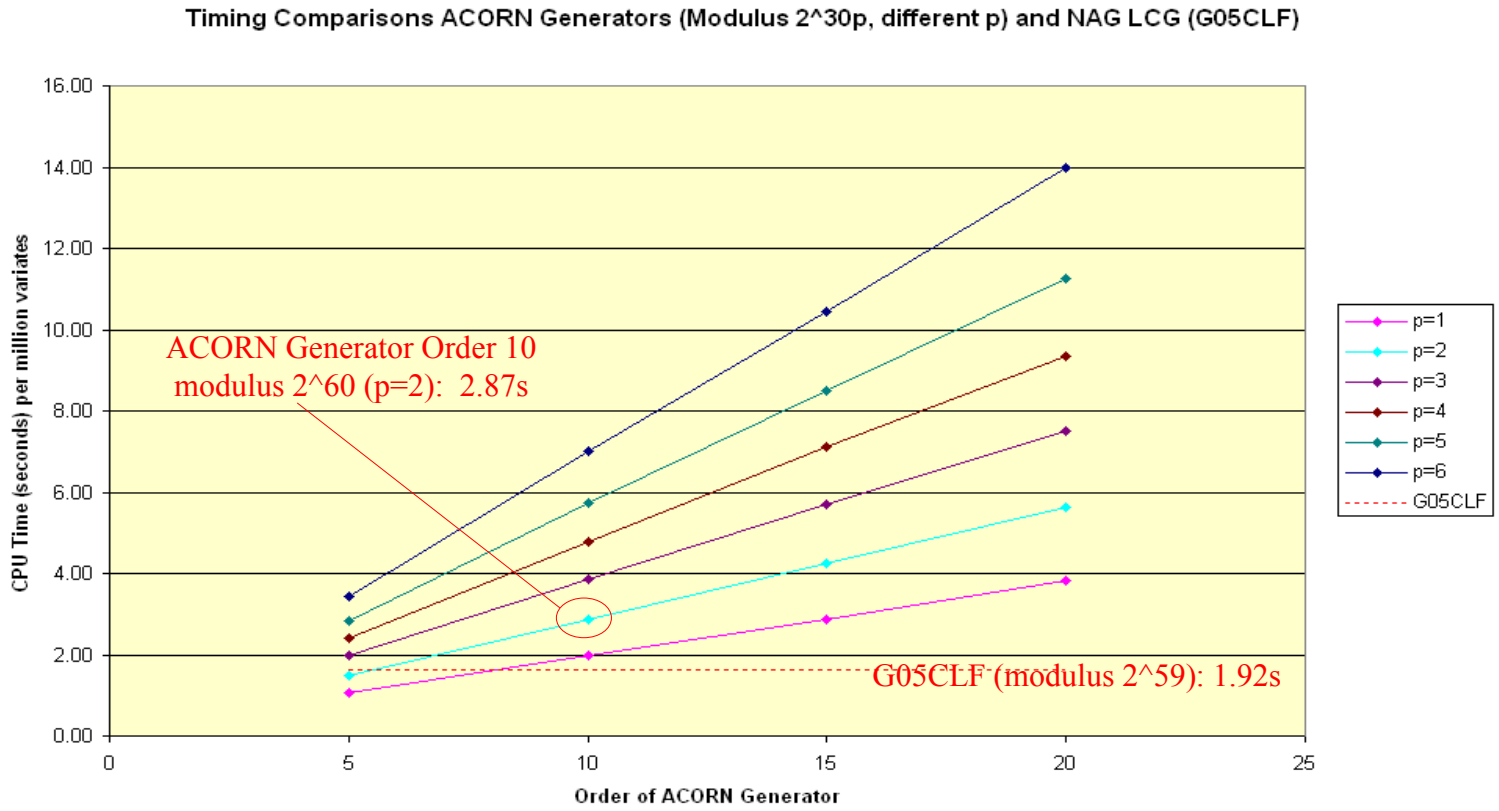
- Modulus 2^{60}
  - Use two integers (I_1, I_2), each less than 2^{30}, to represent a single integer value I less than 2^{60} $\quad I = (2^{30} \times I_1) + I_2$
  - Given two integers I, J represented this way, it is straightforward to do integer addition modulo 2^{60}
- Generalise to modulus 2^{30p}
  - Use p integers (I_1, I_2,…,I_p), each less than 2^{30}, to represent a single integer value I less than 2^{30p}

$$I = (2^{30(p-1)} \times I_1) + (2^{30(p-2)} \times I_2) + \ldots + I_p$$

  - Computational effort to generate each random variate proportional to p (equivalently, proportional to $\log_2 M$)
  - Period length is multiple of the modulus M, as long as seed is odd (ie as long as I_p is odd)

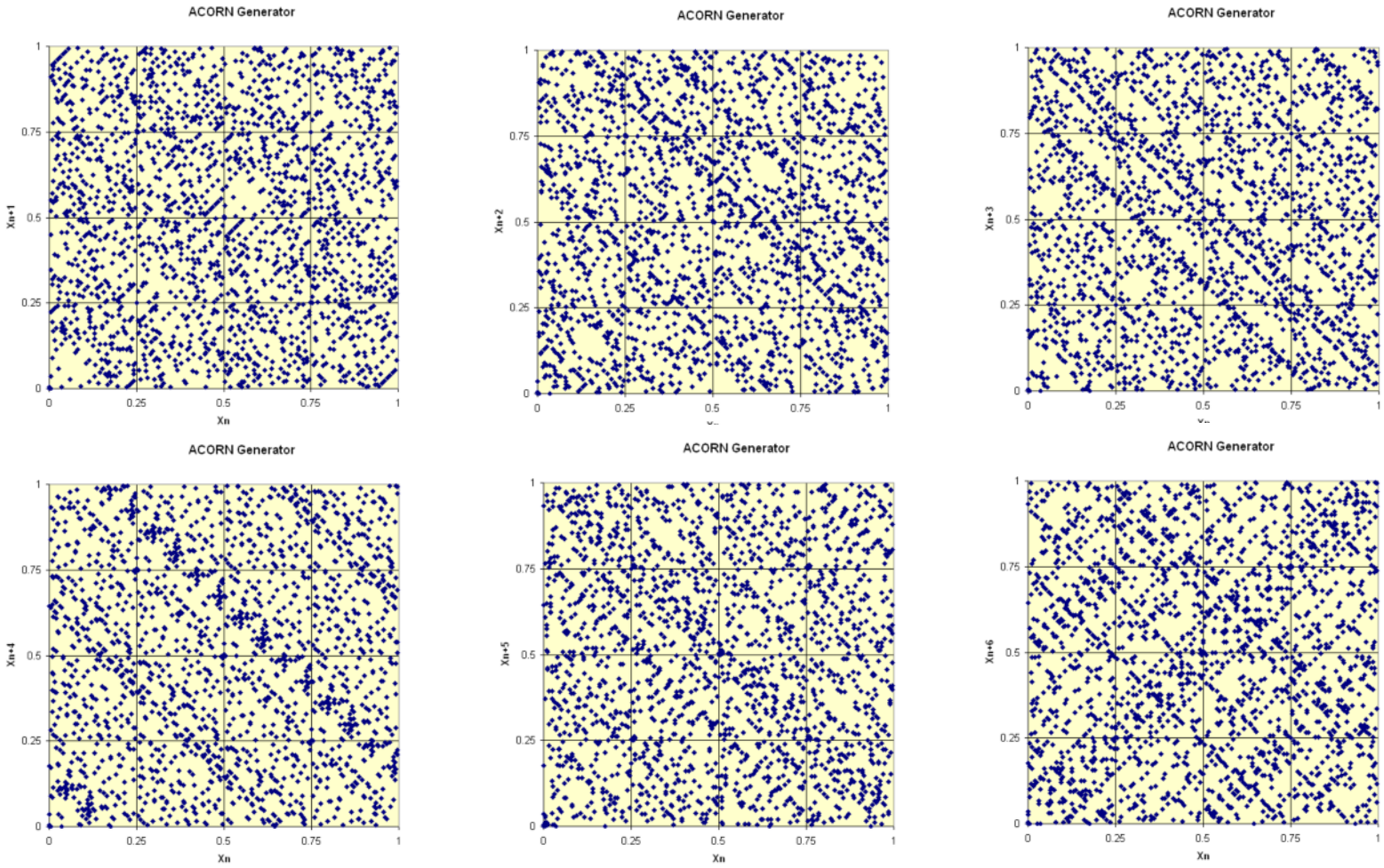
Computational performance (Martyn Byng, NAG)

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

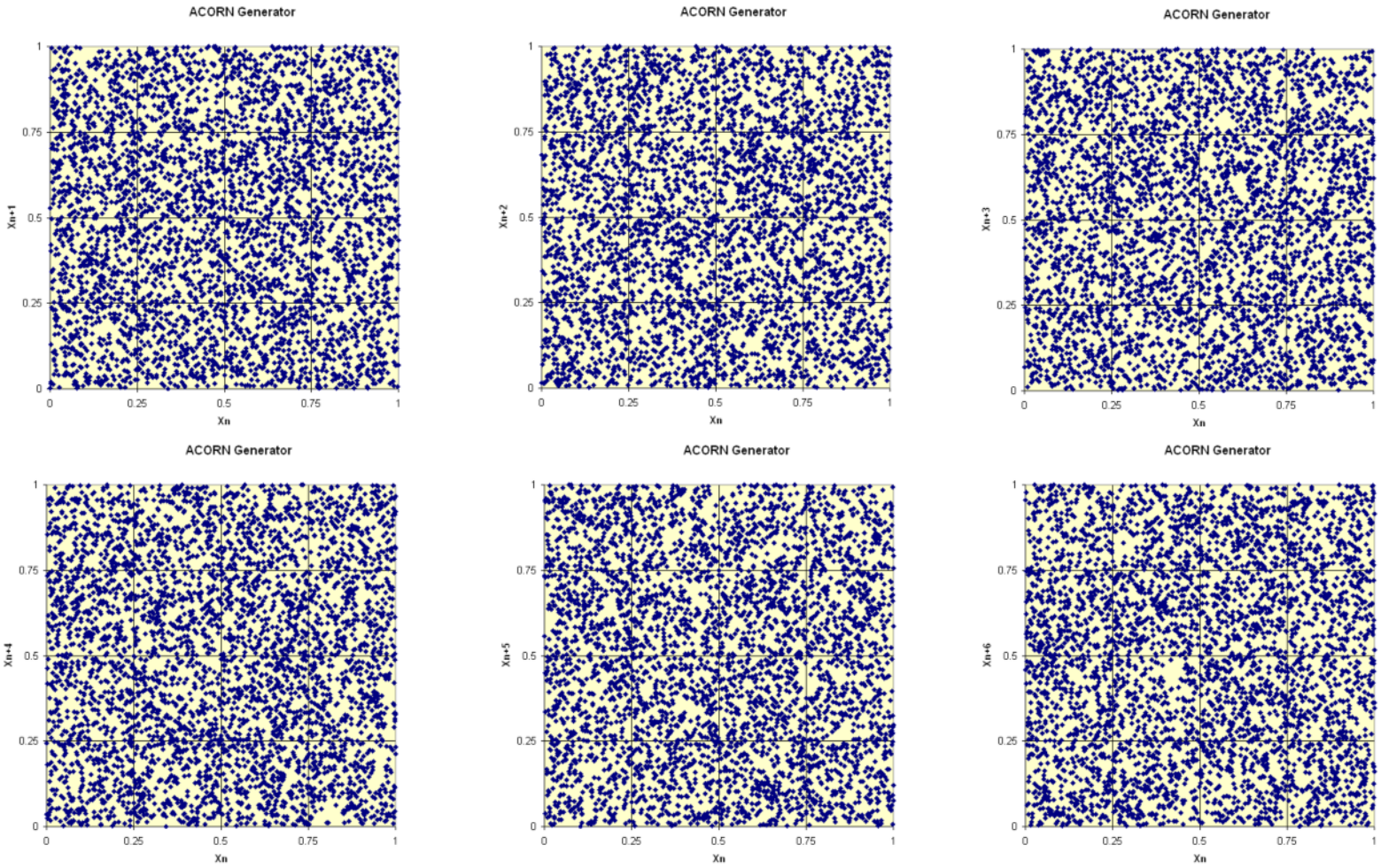


(Timings on:Windows 2000 Professional on Pentium III 600MHz processor with 128Mb memory using Compaq Visual Fortran 6 Compiler)

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



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



RPS Energy

Empirical testing

- Consider ACORN generators with modulus ≥ 2^{60}, order ≥ 10
- Have carried out wide range of tests on the ACORN generators over many years. Some recent tests carried out include
  - Testing on a 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)
  - Application of standard empirical test suites, over last 5+ years
    - Diehard – [Wikramaratna, 2008, submitted to JCAM]
      - Showed that for given modulus, got more bits passing with ACORN than with LCG of same modulus and with good choice of multiplier
      - With modulus ≥ 2^{60}, order ≥ 10 get ~42 random bits (ie full double precision)
    - TestU01 – [Martyn Byng,NAG, personal communication, 2008]

RPS Energy

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
    - STRIDE algorithm
  - Equivalence with specific multiple recursive and matrix generators
  - *k*-th order ACORN generator approximates to *k*-distributed

RPS Energy

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 $Y^m{}_n$

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

RPS Energy

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.

RPS Energy

Conjecture (2007) 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}

RPS Energy

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^2M
…		…		…	…	
$2^i \le k < 2^{i+1}$	$2^i M$	$3^i \le k < 3^{i+1}$	$3^i M$		$q^i \le k < q^{i+1}$	$q^i M$

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
    - Need very long period length (sufficient to carry out full set of realisations)

RPS Energy

STRIDE algorithm

$$\text{Write} \quad Y^{m}{}_{j+n} = \left(\sum_{i=0}^{m} Y^{i}{}_{j} W^{m-i}{}_{n} \right)_{\bmod M} \qquad \text{where} \quad W^{m-i}{}_{n} = \left(Z^{m-i}{}_{n} \right)_{\bmod M}$$

- By calculating the $W^{m-i}{}_{n}$ for a given value of $n=s$ (the initialisation step), it becomes possible to calculate strides of an arbitrary length s through an ACORN sequence (the stride step) by making use of this equation
  - Provided only that it is possible to carry out both multiplication and addition modulo M (note that stride step is carried out once per realisation)
- For initialisation step, an obvious way of calculating the $W^{m-i}{}_{s}$ is to initialise an m-th order ACORN generator with seed equal to 1 and all the initial values zero and apply the ACORN algorithm m times, noting that in this case the $W^{m-i}{}_{n}$ are precisely the $Y^{m-i}{}_{n}$
  - For large s, more efficient approaches to initialisation are possible (but note that in any case the initialisation step only needs carrying out once, or can be pre-calculated)

RPS Energy

STRIDE algorithm (more efficient initialisation)

$$\text{Observe that} \quad W^{m}{}_{2j} = \left(\sum_{i=0}^{m} W^{i}{}_{j} W^{m-i}{}_{j} \right)_{\bmod M}$$

- For large *n* it becomes more efficient to apply an algorithm that takes advantage of this - equivalent to taking stride step of length *j* through ACORN sequence initialised with seed $W^{0}{}_{j}$ and initial values $W^{i}{}_{j}$, i=1, ..., m
- Initialisation for stride length 2^s can be done in time equivalent to *s* 'stride' steps (compared with 2^s 'ACORN' calls using original approach)
- Initialisation for any stride length between 2^s and $(2^{s+1}-1)$ can be done in time equivalent to at most *s* 'STRIDE' steps and *s* 'ACORN' calls

RPS Energy

Equivalence with multiple recursive generators (2007) …

- 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 (2007)

- Generalised MRG can be re-written

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

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

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

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

  - Apply *k* times – gives matrix equation with disjoint vectors
  - This is a particular case of a matrix generator (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 and columns (equivalently, rotating by 180° about mid-point of matrix)

RPS Energy

k-distributed sequences - definitions

- A sequence ($\mathbf{x}_n$) is uniformly distributed modulo 1 in $\mathfrak{R}^k$ if for all [**a**,**b**) contained in or equal to $\mathfrak{I}^k$ (where $\mathfrak{I}^k$ = [**0**,**1**) is unit *k*-cube)

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

- A sequence ($\mathbf{x}_n$) is well distributed modulo 1 in $\mathfrak{R}^k$ 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 $\{\mathbf{x}_{p+n}\}$, $1 \le n < N$ that lie in [**a**,**b**) where {**x**} means fractional parts of **x**
- $A([\mathbf{a},\mathbf{b});N)$ is defined to be equal to $A([\mathbf{a},\mathbf{b});N,0)$

RPS Energy

k-distributed property (1992)

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 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 seeds, this suggests that with large enough modulus ACORN sequences can provide good approximation to *k*-distributed

RPS Energy

k-distributed convergence (2007)

- 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 $b \leq m$; 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 $\chi^0{}_0$ – 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”

RPS Energy

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 in spare time
  - Great opportunity to look at some very interesting applications of mathematics, and to make a real impact
- Currently looking at applications in Monte-Carlo integration
  - Identifying appropriate test examples in higher dimensions
  - Comparing results with different generators
  - Does ‘convergence’ property for ACORN generators give a real practical benefit?
- Numerical Algorithms Group will include ACORN generator in next release of NAG subroutine libraries
  - Provides a focus for more extensive testing and use on wider range of real applications in the future
  - Potential for use in parallel applications (including distributed processing)

RPS Energy

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 any modulus *M* equal to a power of 2
  - Period length is a multiple of the modulus (provided seed and modulus relatively prime) – can be increased without limit
  - Sequences can be reproduced on any machine (to the full 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

References

- ACORN - A New Method for Generating Sequences of Uniformly Distributed Pseudo-random Numbers, *J. Comput. Phys.*, 83 (1989) pp16-31
- 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*, doi: 10.1016/j.cam.2007.05.018, in press (2007)
- Some Convergence Results for Additive Congruential Random Number Generators, (submitted to *J. Comput. and Appl. Mathematics*, July 2007)
- Empirical Testing of the Additive Congruential Random Number Generator, unpublished (submitted to *J. Comput. and Appl. Mathematics*, January 2008)