New Implementations of the Spectral Test

Tim Hopkins
Computing Laboratory, University of Kent
Canterbury, Kent CT2 7NF, UK

July 20, 1997

Abstract

We present three versions of the revised spectral test for the analysis of liner congruential random number generators. One is a Fortran 90 version of the code presented in [6] which extends the range of integer arithmetic operations by performing the arithmetic using floating-point numbers. The range of modulus values which may be analyzed is determined by the length of the mantissa. The other two implementations use the multiple precision arithmetic facilities provided by the Fortran 90 package, mpfun [2] and the Unix program *bc* (a version of this program is freely available from GNU). Both these allow arbitrary values of the modulus to be analyzed notwithstanding the underlying integer and floating-point hardware.

1 Introduction

The spectral test (see [5] for details) analyzes liner congruential random number generators of the form

$$x_{n+1} = (ax_n + c) \bmod m, \qquad n \geq 0$$

Where $c, x_0 > 0$; $0 < a < m$ and a is relatively prime to m. For most useful generators the values of m will be chosen close to the maximum integer value that is storable by the machine. The implementation of the spectral test requires exact integer arithmetic on values up to $4m^2$ which restricts a straightforward Fortran integer implementation to the analysis of m less than the square root of such values. Even careful use of double precision variables to store larger integers, as in [6], does not, in general, provide an adequate range of m values.

Some Fortran 77 compilers allow 16 byte floating-point arithmetic (REAL∗16 or quadruple precision) although this facility does not appear to have been extensively propagated to Fortran 90 compilers. Table 1 indicates the maximum values of m which may be safely analyzed using a number of the most commonly available integer and floating-point representations. Note that, in generating Table 1 we have erred on the side of caution and taken $m_{\max}$ to be $\beta^{\frac{1}{2}n}/8$ where β is the base of the arithmetic and n base-β digits are available for storing the integer data.

Arithmetic	$M_{\max}$
32-bit integer	2^{13}
IEEE double precision	2^{23}
IBM real ∗16	2^{53}
Cray double precision	2^{44}

In §3 we describe a Fortran 90 version of the code which appeared in [6]. This code has a number of advantages over the earlier version; the better control structures available provide more readable code, all the required intrinsic functions are available and the supplementary routine VPROD may be replaced by the new DOT_PRODUCT intrinsic function. Finally, the precision of the floating-point arithmetic used may be changed merely by defining the appropriate KIND value and recompiling.

We then present two implementations using multiple precision arithmetic to ensure that all the integer arithmetic involved is performed exactly for arbitrarily large values of m. The first, described in §4, uses the Fortran 90 *mpfun* package [2]. The second, see §5, is written in *bc*, a widely available facility on Unix machine; a portable version of *bc* is freely available as a GNU package.

In §6 we present some results and compare the execution speed of the codes. Finally in §7 we give details of how the described codes may be obtained.

2 Assessing the Results of the Spectral Test

Details of the theory of the spectral test may be found in [4], [7] and [5]; the latter also contains a detailed derivation of the algorithm.

For given values of a, m and T the codes presented here determine the values of

$$\{\gamma_t, \mu_t \text{ and } \log_2(\gamma_t)\}_{t=2}^{T}$$

such that

1. $$\gamma_t = \min \left\{ \sqrt{\sum_{i=1}^{t} S_i^2} \;\middle|\; \sum_{i=1}^{t} a^{i-1} S_i \equiv 0 \pmod{m} \right\}$$

   where the $\{S_i\}_{i=1}^{t}$ are integers in the range $[0, m)$ and $(S_1 \ldots S_t) \neq (0 \ldots 0)$.

2. $$\mu_t = \frac{\pi^{\frac{t}{2}} \gamma_t^t}{(\frac{t}{2})! m}, \quad \text{where} \quad \left(\frac{t}{2}\right)! = \left(\frac{t}{2}\right)\left(\frac{t}{2} - 1\right) \cdots \left(\frac{1}{2}\right)\sqrt{\pi} \quad \text{for } t \text{ odd}$$

A multiplier a may be considered adequate if the values $\{\mu_t\}_{t=2}^{6}$ all exceed 0.1. For an exceptionally good multiplier, these values will all be greater than unity. Although high values of the μ_t indicate an unusually good multiplier, a, for a given m, in order to ensure that the random numbers generated are also of a high standard it is necessary

1. to check that the values of γ_t are also suitably large. As a guide [5] (p.101) suggests $\log_2(\gamma_t) \geq \frac{30}{t}$, $2 \leq t \leq 6$; and
2. to subject the generator to a number of the empirical tests detailed in [5] (pp.38–71).

Upper bounds on the values of $\{\mu_t\}_{t=2}^{6}$ are given in [5] (pp.103, 105).

It should be noted that the spectral test may also be applied to multiplicative congruential generators provided that

1. m is prime and a has been chosen to produce a period of length $m - 1$; or
2. $m = 2^e$ and a is such that $a \bmod 8 = 5$.

For (i) the test is applied to a and m as given, whereas for (ii) the test is applied to a and $m = 2^{e-2}$ (and in this case a must be less than 2^{e-2})

3 Fortran 90 Floating-point Version

The only major difference between the Fortran implementation given here and the description of the algorithm given in [5] is that the arrays U and V are transposed. This has been done to make the calculation of the inner products required throughout the routine more efficient.

A call to the routine is of the form

```
CALL spect (a, m, outvals, ifault)
```

where

`a` (real, kind = wp), intent (in)
Defines the multiplier of the lcg

`m` (real, kind = wp), intent(in)
Defines the modulus of the lcg.

`outvals` (array of type `out_vals` with range (2..T)), intent (out)

Contains the values of γ_t^2, $\log_2 \gamma_t$ and μ_t for $t = 2 \ldots T$. The routine determines the value of T required from the upper bound of this array.

The type `out_vals` is defined as

```
TYPE out_vals
    REAL (kind=wp): nu, log_nu, mu
END TYPE out_vals
```

`ifault` (integer), intent (out)

Indicates the error status of the routine on exit

0	–	routine executed without error
1	–	lower bound of `outvals` > 2 or upper bound < 2
2	–	$a \geq m$ or $a \leq 0$ or $m \leq 0$
3	–	$m > m_{\max}$
4	–	a and m not relatively prime
5	–	intermediate result $> m_{\max}^2$

The following example driver program shows the routine being used to analyze the generator defined by $a = 137$ and $m = 256$ ([5], p.102 line 5).

```
    PROGRAM test
      USE spectral_test, ONLY : wp, spect, out_vals
      INTEGER bigt, outch
      PARAMETER (bigt=6,outch=6)
      REAL (wp) :: a, m
      TYPE (out_vals) :: outvals(bigt)
      INTEGER ifault, i
!
!..Example program illustrating the use of subroutine spect
!..to rate linear congruential generators of the form
!..          X(I+1) = (A*X(I) + C) MOD M
!..for  a=137; m=256; Knuth p102 no 5
!
      WRITE (outch,"(' **** example program for spect ****')")
      WRITE (outch,"(/' Knuth page 102 number 5')")
      a = 137D0
```

```
      m = 256D0
      WRITE (outch, &
  "(/' Multiplier= ',f12.0,12x,'modulus= ',f12.0/)") a, m
      CALL spect(a,m,outvals,ifault)
      IF (ifault==0) THEN
!  OUTPUT RESULTS
        WRITE (outch,"('  t',8x,'nusq(i)',3x,'lognu(i)', &
       5x,'mu(i)')")
        DO i = 2, bigt
          WRITE (outch,"(1X,I2,2X,F12.0,4X,F6.2,6X,F5.2)") &
            i, outvals(i)%nusq, outvals(i)%lognu, outvals(i)%mu
        END DO
      ELSE
        CALL moan(ifault,outch)
      END IF
    END
    SUBROUTINE moan(ifault,outch)
      INTEGER ifault, outch
!  FAULT INDICATOR ROUTINE
        SELECT CASE (ifault)
        CASE(1)
          WRITE (outch, &
           "(' Outvals: Upper bound <2 or lower bound >2 on entry')")
        CASE(2)
          WRITE (outch, &
   "(' a.ge.m or a.le.0 or m.lt.0 on entry')")
        CASE(3)
          WRITE (outch,"(' m.gt.mmax on entry')")
        CASE(4)
          WRITE (outch,"(' a and m are not relatively prime')")
        CASE(5)
          WRITE (outch, &
   "(' internally produced value .gt. mmax, please report')")
        END SELECT
    END
```

The `USE spectral_test` statement is required to make available the kind value of the floating-point arithmetic being used to perform the integer calculations and the definition of the type `out_vals`.
The following output is produced

```
 **** example program for spect ****

 Knuth page 102 number 5

 Multiplier=         137.            modulus=         256.

  t        nusq(i)   lognu(i)     mu(i)
  2          274.      4.05       3.36
  3           30.      2.45       2.69
  4           14.      1.90       3.78
  5            6.      1.29       1.81
  6            4.      1.00       1.29
```

4 Fortran 90 Multiple Precision Version

The Fortran 90 multiple precision version uses the `mpfun` library [2] of routines. The library implements the basic arithmetic operations on multiple precision integers and reals by overloading the existing operators; it also provides a number of the more commonly used intrinsic functions. Use of this package results in code that is far more readable than that resulting from the use of earlier libraries (for example, [1] and [3]) where each arithmetic operation requires a function call.

All calculations, except the computation of the μ_t and $\log_2(\gamma_t)$, are performed using `mp_integer` data. The use of the `INT` intrinsic function is unnecessary since the floor function is obtained through integer division.

A call to the routine is of the form

```
CALL mp_spect (mp_a, mp_m, mp_outvals, ifault)
```

where

`mp_a` (mp_integer), intent(in)
Defines the multiplier of the lcg.

`mp_m` (mp_integer), intent(in)
Defines the modulus of the lcg

`mp_outvals` (array of type `mp_out_vals` with range 2..T)), intent (out)

Contains the values of γ_t^2,$\log_2 \gamma_t$ and μ_t for $t = 2 \ldots T$. The routine determines the value of T required from the upper bound of this array.

The type `mp_out_vals` is defined as

```
TYPE mp_out_vals
    TYPE (mp_integer) nusq
    REAL (wp) mu, lognu
END TYPE mp_out_vals
```

`ifault` (integer), intent (out)

Indicates the error status of the routine on exit

0	–	routine executed without error
1	–	lower bound of `mp_outvals` > 2 or upper bound < 2
2	–	$a \geq m$ or $a \leq 0$ or $m \leq 0$
3	–	a and m not relatively prime
4	–	unable to allocate internal workspace
5	–	unable to deallocate internal workspace (this is a warning message; the returned results are correct).

The following example program illustrates the routine being called to analyze the generator defined by $a = 137$ and $m = 256$ ([5], p.102 line 5).

```
PROGRAM test
  USE mpmodule
  USE mp_spectral_test
  INTEGER bigt, outch
  PARAMETER (bigt=6,outch=6)
  TYPE (mp_out_vals) :: outvals(bigt)
  TYPE (mp_integer) a, m
  INTEGER ifault, i
!
```

```
!..Example program illustrating the use of subroutine mp_spect
!..to rate linear congruential generators of the form
!..          X(I+1) = (A*X(I) + C) MOD M
!..
!..for a=137; m=256; Knuth p102 no 5
!
      CALL mpinit
      WRITE (outch,"(' **** Example program for spect ****')")
      WRITE (outch,"(/'Knuth page 102 number 5')")
      a = 137
      m = 256
      WRITE (outch,"(' Multiplier')")
      CALL mpwrite(outch,a)
      WRITE (outch,"(' Modulus')")
      CALL mpwrite(outch,m)
      WRITE (outch,"()")

      CALL mp_spect(a,m,outvals,ifault)
      IF (ifault==0) THEN
!..Output results
        WRITE (outch,"('  t    lognu(t)     mu(t)')")
        DO i = 2, bigt
          WRITE (outch,"(1X,I2,4X,F6.2,6X,F5.2)") &
    i, outvals(i) %lognu, outvals(i) %mu
        END DO
        WRITE (outch,"(1X,/'Nusq values')")
        DO i = 2, bigt
          CALL mpwrite(outch,outvals(i)%nusq)
        END DO
      ELSE
        CALL moan(ifault,outch)
      END IF
    END
    SUBROUTINE moan(ifault,outch)
      INTEGER ifault, outch
!..Fault indicator routine
      SELECT CASE (ifault)
      CASE(1)
        WRITE (outch, &
          "(' Outvals: Upper bound <2 or lower bound >2 on entry')")
      CASE(2)
        WRITE (outch,"(' a.ge.m or a.le.0 or m.lt.0 on entry')")
      CASE(3)
        WRITE (outch,"(' a and m are not relatively prime')")
      CASE(4)
        WRITE (outch,"(' Unable to allocate internal workspace')")
      END SELECT
    END
```

Notes

1. The `USE mpmodule` statement is required to make the definitions of the type `mp_integer` visible to the calling program.

2. The `USE mp_spectral_test` statement makes the definition of `mp_outvals` visible to the calling program.

3. The call to the routine `mpinit` is needed to initialize the `mpfun` package.

4. There is little control available over the format of the output multiple precision values.

The following output is produced

```
 **** Example program for spect ****

Knuth page 102 number 5
 Multiplier
10 ^         2 x  1.37,
 Modulus
10 ^         2 x  2.56,

  t    lognu(t)     mu(t)
  2      4.05       3.36
  3      2.45       2.69
  4      1.90       3.78
  5      1.29       1.81
  6      1.00       1.29

Nusq values
10 ^         2 x  2.74,
10 ^         1 x  3.,
10 ^         1 x  1.4,
10 ^         0 x  6.,
10 ^         0 x  4.,
```

5 bc Implementation

bc is an interactive program. The spectral test needs to be preloaded before the spectral test function k, may be called. A call is of the form $k(a, m, t)$ where t is the maximum value to t required. Assuming that the *bc* version of the spectral test has been placed in the file *spect.b* the example linear congruential generator $a = 137$ and $, = 256$ ([5], p.102 line 5) may be analyzed using

```
% bc spect.b
k(137,256,6)
```

which gives

```
a= 137
m= 256
i= 2
mu[i]= 3.36248588704532557557
nusq[i]= 274
i= 3
mu[i]= 2.68862681697444208212
nusq[i]= 30
i= 4
mu[i]= 3.77820793479202009634
nusq[i]= 14
i= 5
mu[i]= 1.81316210593470716753
nusq[i]= 6
```

```
i= 6
mu[i]= 1.29192819501249250735
nusq[i]= 4
```

6 Results

The codes have been tested on a wide range of examples. Included with the distribution are data and results files for both the multiple precision codes which generate all the example values given in the table on pp.101–102 of [5]. Because of the restrictions placed on $m_{\max}$ by the double precision version of the code only a relatively small number of generators have been tested.

In running all the example in the table in [5] the gnu version of *bc* ran approximately twice as fast as the version bundled with SunOS (141 seconds as against 282 seconds on a SUN LX running SunOS 4.3). The implementation using the `mpfun` library takes 82 seconds using the Edinburgh Portable Compilers (EPC) Fortran 90 compiler; this timing was achieved with unoptimized code, bugs in the optimizer preclude its use on the `mpfun` package. All times include I/O.

7 Obtaining the Code

A compressed, tar file containing

- the three versions of the spectral test described in this paper along with example programs, stringent test drivers and data,
- the version of the mpfun library used,

may be obtained via anonymous ftp from unix.hensa.ac.uk from the file

```
/pub/misc/trh/spectral/test.tar.gz
```

Once retrived the file should be uncompressed using

```
%tar xvf test.tar
```

The gnu version of bc may be obtained via anonymous ftp from either *uu.net* in the file

```
ftp://ftp.uu.net/systems/gnu/bc-1.03.tar.gz
```

or from *unix.hensa.ac.uk* in the file

```
ftp://unix.hensa.ac.uk/mirrors/gnu/bc-1.03.tar.gz
```

A makefile is included that will

- compile and run the example Fortran 90 double precision version (make test_f90),
- compile the `mpfun` library and check the implementation (make test_mpfun),
- compile and run the example and stringent test drivers for the multiple precision version (make test_mpspec and make test_mpknuth).

The makefile is very crude. Since all current Fortran 90 systems seem to have their own idiosyncratic means of linking precompiled modules with driver programs, we just compile all the required source files each time. It would obviously be far more efficient to precompile the multiple precision arithmetic library and link this with each of the test drivers.

The gnu version version of *bc* is a complete rewrite and allows a number of extensions to the standard version (for example, longer than a single character identifier names). None of these extensions have been used in the code provided. The gnu version is also substantially faster than its standard counterparts.

Please report any problems, bugs or possible improvements to trh@ukc.ac.uk

References

[1] D.H. Bailey. Multiprecision translation and execution of Fortran programs. *ACM Trans. Math. Softw.*, 19(3):288–319, Sep 1993.

[2] D.H. Bailey. A Fortran 90-based multiprecision system. *ACM Trans. Math. Softw.*, 21(4):379–387, Dec 1995.

[3] R.P. Brent. MP: a Fortran multiple-precision arithmetic package. *ACM Trans. Math. Softw*, 4(1):71–81, Mar 1978.

[4] R.R. Coveyou and R.D. MacPherson. Fourier analysis of uniform random generators. *J. Ass. Comput. Mach.*, 14:100–119, 1967.

[5] D.E.Knuth. *The Art of Computer Programming Volume 2: Seminumerical Algorithms*. Addison-Wesley, Reading, Mass., 2nd edition, 1981.

[6] Tim Hopkins. Algorithm as 183: A revised algorithm for the spectral test. *Applied Statistics*, 32(3):328–335, 1983.

[7] G. Marsaglia. Random numbers fall mainly in the planes. *Proc. Nat. Acad. Sci. USA*, 61:25–28, 1968.