Unclassified
PRTD(90)R62

THEORETICAL ANALYSIS OF THE ACORN RANDOM NUMBER GENERATOR

ROY S WIKRAMARATNA

(Paper presented at the SIAM Conference on Applied Probability in Science and Engineering, New Orleans, Louisiana, March 5-7 1990)

Petroleum Reservoir Technology Division
Winfrith Petroleum Technology
Dorchester
Dorset DT2 8DH
United Kingdom

March 1990

Short Title: Analysis of ACORN Generator

CONTENTS

ABSTRACT

The ACORN (additive congruential random number) generator has been proposed by Wikramaratna (J.Comput. Phys., 83(1)16-31, 1989) as a source of uniformly distributed pseudo-random numbers. The k-th order ACORN generator is defined recursively from a seed and k initial values, leading to an algorithm which is extremely simple to implement. Further advantages of the algorithm include long period length and speed of execution.

In this work we investigate the theoretical background for the ACORN generators and present some preliminary results of this investigation. We derive explicit representations for the terms in the ACORN sequences and show how this representation can be used to model the divergence of neighbouring orbits resulting from small perturbations in either the seed or the initial values. We use this model to illustrate the benefits of coding the algorithm in integer arithmetic modulo M rather than in real arithmetic modulo 1.

1. INTRODUCTION

In a recent paper, Wikramaratna [1] proposed the ACORN (Additive Congruential Random Number) generators as a source of uncorrelated random numbers uniformly distributed in the unit interval. The k-th order ACORN generator is defined recursively from a seed X^0_0 $(0 < X^0_0 < 1)$ and a set of k initial values X^m_0, $m = 1, \ldots, k$ each satisfying $0 \le X^m_0 < 1$ by:

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

$$X^m_n = (X^{m-1}_n + X^m_{n-1})_{\text{mod } 1} \qquad n \ge 1,\ m = 1, \ldots k \tag{2}$$

where $(X)_{\text{mod } 1}$ means the fractional part of X. Alternatively, using integer arithmetic, it can be defined from a modulus M, a seed Y^0_0 $(0 < Y^0_0 < M)$ and initial values Y^m_0, $m = 1, \ldots, k$ satisfying $0 \le Y^m_0 < M$ by:

$$Y^0_n = Y^0_{n-1} \qquad n \ge 1 \tag{3}$$

$$Y^m_n = (Y^{m-1}_n + Y^m_{n-1})_{\text{mod } M} \qquad n \ge 1,\ m = 1, \ldots, k \tag{4}$$

$$X^m_n = Y^m_n/M \qquad m = 1, \ldots, k \tag{5}$$

where M is a suitable large integer and where $(Y)_{\text{mod } M}$ means the remainder on dividing Y by M. It is worth noting that, for the case where all the X^i_0 are rational fractions, the two definitions are exactly equivalent (suppose that $X^i{}_0 = N^i/D^i$, $i=0,\ldots,m$ where the N^i and D^i are integers, then choose M as the lowest common multiple of the D^i, and let $Y^i{}_0 = MX^i{}_0$).

Either definition can be used successfully as the basis for an implementation of the ACORN algorithm; the example listing in reference [1] was based on equations (1) and (2), but it is a simple matter to modify this to implement equations (3)-(5), as in the example given in Figure 1. This algorithm has been implemented for $M = 2^{30}$, and the statistical tests from [1] applied to the resulting sequences of numbers; the results obtained for the tests are very similar to those obtained with the original implementation. There are a number of practical advantages in using the second definition in implementing the generator:

(i) At each stage all of the $Y^m{}_n$ are calculated exactly, so that the values of $Y^m{}_n$ will be the same on any machine provided only that the seed, the initial values and the modulus M are chosen to have the same values. The values of $X^m{}_n$ calculated from (5) may differ slightly due to variations in the machine representations of real numbers, but these differences will only appear in the least significant digits and there will be no cumulative buildup or growth of these differences.

(ii) By contrast, any small differences in machine representations of the seed or initial values will grow rapidly in the algorithm defined by (1) and (2), so that in general it will not be possible to generate the same sequences on different machines, even with the 'same' seed and initial values. We will demonstrate below how quickly such differences can grow. This fact does not invalidate the use of the algorithm as a source of random numbers - although the sequences may differ, they exhibit the same statistical properties - but if it is desirable to repeat a computation on a different machine with the same sequence of random numbers, then the algorithm should be implemented using (3) - (5).

(iii) The theoretical results which were proven in [1] concerning the period length of the resulting sequences hold for the algorithm defined by (3) - (5). For the implementation in real arithmetic, these theoretical results can be used to give an order of magnitude only, and no precise results can be obtained.

(iv) The apparent limitation of $M \leq 2^{30}$ for an implementation in integer arithmetic on a 32-bit machine can be easily overcome, by using two (or more) words to represent each integer $Y^m{}_n$ although this will obviously require greater computational effort. An implementation using p 32-bit integers to represent each Y^m_n permits a choice of modulus up to a maximum of 2^{30p}, and it is thus possible to generate arbitrarily long sequences by a suitable choice of p.

In this paper we derive some preliminary theoretical results concerning the behaviour of the ACORN sequences; these results permit us to analyse the behaviour of the ACORN sequences in response to small perturbations to the seed or the initial values, and lead to a better understanding of the algorithm. They also indicate the potential for further analysis of the ACORN algorithm, and the possibility of proving a priori results concerning the randomness of the resulting sequences of numbers.

2. <u>EXPLICIT REPRESENTATION OF Y^m_n</u>

The algorithm defined by (3) - (5) is very well suited to the computation of the Y^m_n. However it is useful in understanding the behaviour of the ACORN generators to derive explicit representations for the Y^m_n in terms of the modulus, seed and initial values.

Consider first the special case where the seed is equal to one and the initial values are all equal to zero. The resulting sequence of numbers will be denoted by Z^m_n. Making use of a standard result (see for example Prudnikov et al [2], equation 4.1.1.26).

$$\sum_{i=1}^{n} i\,(i+1)\,\ldots\,(i+m) = n(n+1)\,\ldots\,(n+m+1)/(m+2) \qquad (6)$$

equations (3) and (4) can be rewritten for this special case as

$$Z^0_n = 1$$

$$Z^1_n = n$$

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

$$Z^3_n = \sum_{i=1}^{n} i(i+1)/2 = n(n+1)(n+2)/3!$$

$$\vdots$$

$$Z^m_n = \sum_{i=1}^{n} i(i+1) \ldots (i+m-2)/(m-1)!$$

$$= n(n+1) \ldots (n+m-1)/m! \qquad (7)$$

$$= (n+m-1)!/\left[(n-1)!m!\right]$$

The equations (7) hold as long as the right hand side is smaller than M. Once it exceeds this value, then the correct result is still obtained from equation (7) provided that the value is taken modulo M.

Making use of the additive nature of the ACORN generator, it is now trivial to show that for the general case of $1 \le Y^0_0 < M$ and $0 \le Y^m_0 < M$, m=1, ..., k the following equation holds

$$Y^m_n = \left[\sum_{i=0}^{m} Y^i_0 \, Z^{m-i}_n \right]_{\text{mod } M} \qquad (8)$$

where the Z^{m-i}_{n} are as defined by equation (7).

Applying a similar analysis to the sequence defined by equations (1) and (2) leads to an analgous equation defining X^m_n for the general case $0 < X^0_0 < 1$ and $0 \leq X^m_0 < 1, m=1, \ldots, k$.

Thus

$$X^m_n = \left(\sum_{i=0}^{m} X^i_0 \, Z^{m-i}_n \right)_{\bmod 1} \tag{9}$$

where the Z^{m-i}_{n} are as before.

3. <u>DIVERGENCE OF NEIGHBOURING ORBITS</u>

Consider the sequence Y^k_n defined by the seed Y^0_0 and initial values Y^m_0, $m=1, \ldots, k$. Suppose that $\tilde{Y}^k_n$ is the perturbed sequence obtained by changing either the seed or one of the initial values by the smallest possible pertubation; thus $\tilde{Y}^m_0 = Y^m_0$, $m=0, \ldots, k$ $(m \neq i)$ and $\tilde{Y}^i_0 = Y^i_0 + 1$. Now making use of the linearity of (8), an expression for the difference between the two sequences is given by

$$\varepsilon^k_n = \left(Z^{k-i}_n \right)_{\bmod M} \tag{10}$$

The two sequences will be said to have <u>diverged</u> once there is no longer any discernible relationship between them; this will certainly be the case once

$$Z^{k-i}_{n} > M/2 \tag{11}$$

Figure 2 shows values of Z^m_n plotted against n on a log-log scale for a series of values of $m=(k-i)$. Given values of M and m, it is possible to determine approximately the smallest value n for which (11) holds by drawing a horizontal line from the value M/2 on the y-axis until it intersects with the curve for the appropriate value of m; if the x-coordinate of this point of intersection is x_D, then equation (11) holds for all integers $n > x_D$ and if n_D is the smallest such integer the two sequences will have diverged after n_D terms. However, it is difficult to obtain an accurate estimate of n_D by this method, particularly for larger values of M and m.

It is not easy to calculate n_D directly from (7) and (11). However, making use of the inequality

$$\begin{aligned}\left[(n+(m-1)/2)^2 - ((m-1)/2)^2\right]^{m/2} &= \left[n(n+m-1)\right]^{m/2} \\ &\le n(n+1)\ \dots\ (n+m-1) \\ &\le (n+(m-1)/2)^m \qquad m \ge 1,\ n \ge 1 \end{aligned} \tag{12}$$

it is possible to obtain upper and lower bounds for n_D.

Thus,

$$\begin{aligned} M/2 \le Z^{k-1}_{n_D} &= n_D\ (n_D+1)\ \dots\ (n_D+m-1)/m! \\ &\le (n_D+(m-1)/2)^m/m! \end{aligned} \tag{13}$$

$$n_D \geq (m!\ M/2)^{1/m} - (m-1)/2 \tag{14}$$

and

$$M/2 \geq Z^{k-1}_{n_D} - 1 = (n_D-1)\ n_D \ldots (n_D+m-2)/m!$$

$$\geq \left[\left((n_D-1) + (m-1)/2\right)^2 - \left((m-1)/2\right)^2\right]^{m/2}/m! \tag{15}$$

$$n_D - 1 \leq \left[\left(m!\ M/2\right)^{2/m} + \left((m-1)/2\right)^2\right]^{1/2} - (m-1)/2$$

$$\leq (m!\ M/2)^{1/m} \tag{16}$$

Combining (14) and (16)

$$(m!\ M/2)^{1/m} - (m-1)/2 = \underline{n}_D \leq n_D$$

$$\leq \bar{n}_D = (m!\ M/2)^{1/m} + 1 \tag{17}$$

Thus n_D can be bounded below and above by $\underline{n}_D$ and $\bar{n}_D$ which can each be determined directly given the values of M and m. Further, from (16) it follows that

$$\bar{n}_D - \underline{n}_D = 1 + (m-1)/2$$

$$= (m+1)/2 \tag{18}$$

so that the estimate $\bar{n}_D$ will overestimate n_D by at most $(m+1)/2$. Table I compares values of the calculated bounds $\underline{n}_D$ and $\bar{n}_D$ with the actual values obtained for n_D for a series of values of m and M. It will be observed from the table that the lower bound is in practice very close to the actual value. Give M and $m = k-i$, a practical method for obtaining the exact value for n_D is to evaluate $\underline{n}_D$ and $\bar{n}_D$ from (17), then evaluate $Z^{k-i}{}_n$ from (7) for integer values of n between $\underline{n}_D$ and $\bar{n}_D$; n_D is then the smallest value of n for which equation (11) is satisfied.

4. GROWTH OF ROUNDING ERRORS

Suppose that the k-th order ACORN generator is implemented in real arithmetic, for example as in the listing given by Wikramaratna [1]. Small differences in arithmetic rounding may lead to a difference in the precise value of the seed - this may be due to a number of causes, including

(i) Differences in the machine representation of real numbers on different machines

(ii) Differences between compilers

(iii) Use of different levels of optimisation with the same compiler

(iv) Minor changes to the code, resulting in the calculation of the seed being undertaken in a slightly different way.

Suppose then that we make a small perturbation of size ε in the seed $X^0{}_0$. The expressions derived in section 2 allow us to make an estimate of the rate of growth of this perturbation, and illustrate the importance of using the implementation in integer arithmetic if the sequences are required to be reproducible at a future date. The growth of the perturbation can be modelled by $\varepsilon Z^k{}_n$, that is by the growth of a unit perturbation in an integer implementation of the k-th order ACORN generator with $M = \varepsilon^{-1}$. Similarly, the growth of a perturbation of size ε in the i-th initial value $X^i{}_0$ can be modelled by $\varepsilon Z^{k-i}{}_n$.

This is illustrated in single precision real arithmetic for the case of a minor change to the calculation of the seed, replacing a calculation of the form

```
XV(1) = 0.1 * SQRT (2.0)/1.42
```

by the calculation

```
XV(1) = SQRT (2.0)/14.2
```

The only difference in the two values of the seed XV(1) is the rounding error induced by the different order of the computation. Two sequences A_n and B_n of random numbers were generated for the 10-th order ACORN generator using the two methods of calculating the seed given above. The initial difference in the seeds was 7.5×10^{-9}, approximately 2^{-27}. In each case the initial values were all identically zero. Let δ_n be the difference modulo 1 between the n-th members of the two sequences, thus

$$\delta_n = (A_n - B_n)_{\mathrm{mod}\ 1}$$

$$1-\delta_n = (B_n - A_n)_{\mathrm{mod}\ 1} \qquad (19)$$

By definition, both δ_n and $(1-\delta_n)$ must always be less than or equal to one. Of the sequences δ_n and $(1-\delta_n)$, one will start close to zero and increase slowly at first and then more rapidly towards the value one, after which there is no longer any discernible pattern to the values; the other will start close to one and decrease towards zero.

Figure 3a shows values of δ_n and $(1-\delta n)$ plotted against n on a linear scale. In particular it should be noted that the two sequences have diverged at the point where the plots of δ_n and $(1-\delta_n)$ intersect (ie after the first 23 terms in the sequences have been generated).

Figure 3b shows the growth of a perturbation of size 1 in the 10th order ACORN generator, implemented in integer arithmetic with $M=2^{27}$; The plot shows values of $Z^{10}{}_n/M$ plotted against n. These values could be obtained by calculating $Z^{10}{}_n$ from equation (7) and normalising to the unit interval; in practice it is simpler to make a series of calls to the Fortran function ACORNI (listed in figure 1) with order 10, modulus 2^{27}, seed 1 and initial values all zero which gives the values of $Z^{10}{}_n/M$ directly. The model predicts divergence of the sequences after 24 terms.

Finally figure 3c shows the ratio between the rounding error δ_n and the modelled pertubation growth $Z^{10}{}_n/M$, plotted against n; this ratio is close to one (fluctuating between 0.8 and 1.4), which further demonstrates the adequacy of the model for the growth of the rounding errors up to the point at which the sequences have diverged.

In double precision arithmetic, the initial rounding error is much smaller, and the sequences diverge more slowly; however the 10th order sequences still diverge after a few hundred terms (as compared with a few tens of terms in single precision arithmetic).

5. CONCLUSIONS

(i) There are a number of advantages of implementing the ACORN algorithm in exact integer arithmetic, equations (3) - (5), rather than using real arithmetic, equations (1) - (2), as proposed originally by Wikramaratna [1]. The main advantage is that the same sequence can be reproduced on any machine. A secondary advantage is that the integer algorithm is more amenable to theoretical analysis, as demonstrated both in this paper and in [1]. However, if reproducibility is not important, the real implementation is perfectly adequate in practice, as shown by the results of statistical tests applied to the sequences which are generated [1].

(ii) In this paper we have derived explicit formulae for the numbers generated by the k-th order ACORN generators implemented in exact integer arithmetic for any values of the modulus, seed and initial values. These formulae improve our understanding of the behaviour of the ACORN sequences and allow us to analyse the divergence of neighbouring orbits in the integer implementation. For an implementation in real arithmetic, they provide an approximate model for the rate of growth of rounding errors.

(iii) The explicit formulae will provide a basis for further theoretical analysis of the ACORN generator, aimed at proving a priori results concerning the randomness of the resulting sequences of numbers.

6. REFERENCES

[1] R.S.Wikramaratna, ACORN - A New Method for Generating Sequences of Uniformly Distributed Pseudo-random Numbers, J Comput. Phys., 83(1)16-31 (1989)

[2] A.P.Prudnikov, Yu.A.Brychkov and O.I.Marichev, Integrals and Series - Volume I Elementary Functions (Gordon and Breach Science Publishers, New York, 1986).

```
      DOUBLE PRECISION FUNCTION ACORNI(XDUMMY)
C
C         Fortran implementation of ACORN random number generator
C         of order less than or equal to 12 (higher orders can be
C         obtained by increasing the parameter value MAXORD).
C
C         R.S.Wikramaratna
C         Winfrith Petroleum Technology
C         Dorchester, Dorset DT2 8DH, United Kingdom.
C
C         The variable XDUMMY is a dummy variable. The common block
C         IACO is used to transfer data into the function.
C
C         Before the first call to ACORN the common block IACO must
C         be initialised by the user, as follows. The values of
C         variables in the common block must not subsequently be
C         changed by the user.
C
C         KORDEI    - order of generator required ( must be =< MAXORD)
C
C         MAXINT    - modulus for generator, must be chosen small
C                     enough that 2*MAXINT does not overflow
C
C         IXV(1)    - seed for random number generator
C                     require 0 < IXV(1) < MAXINT
C
C         (IXV(I+1),I=1,KORDEI)
C                   - KORDEI initial values for generator
C                     require 0 =< IXV(I+1) < MAXINT
C
C         After initialisation, each call to ACORN generates a single
C         random number between 0 and 1.
C
C         An example of suitable values for parameters is
C
C           KORDEI   = 10
C           MAXINT   = 2**30
C           IXV(1)   = an odd integer in the (approximate) range
C                      (0.001 * MAXINT) to (0.999 * MAXINT)
C           IXV(I+1) = 0, I=1,KORDEI
C
      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) IXV(I+1)=IXV(I+1)-MAXINT
    7 CONTINUE
      ACORNI=DFLOAT(IXV(KORDEI+1))/MAXINT
      RETURN
      END
```

FIG. 1. Fortran implementation of the ACORN generator in integer arithmetic

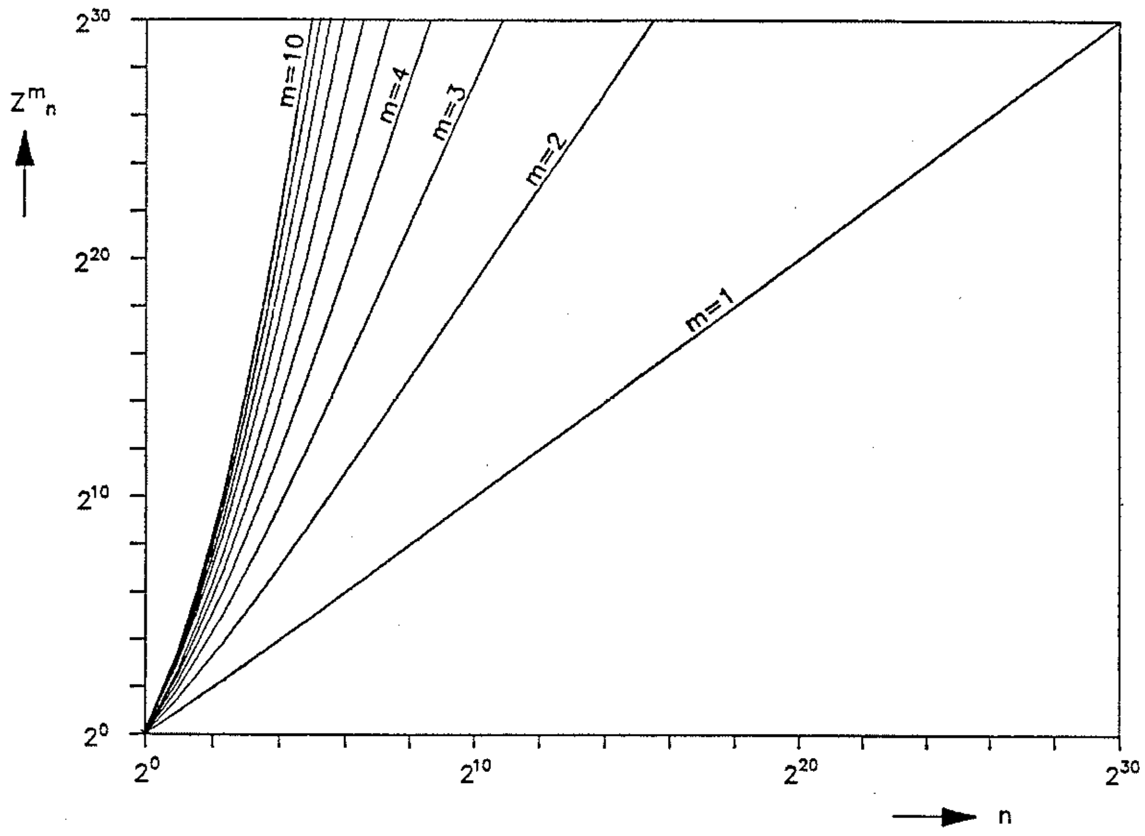

FIG. 2. Graph showing Z^m_n plotted against n for values of m between 1 and 10

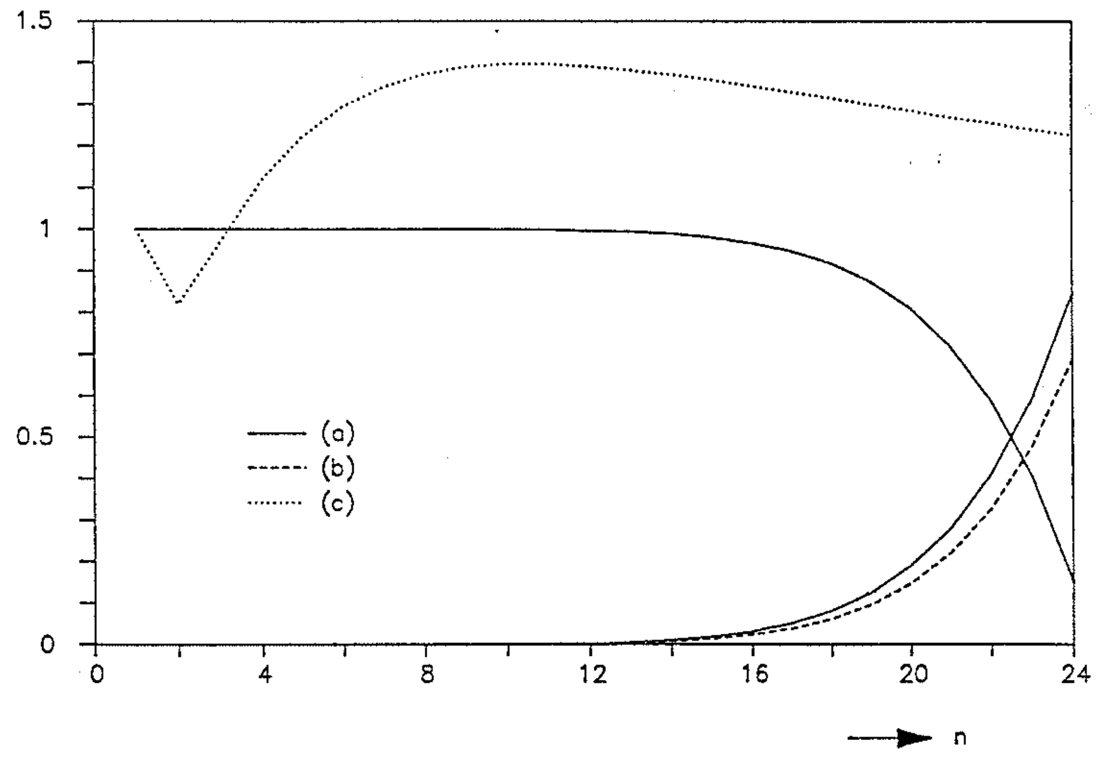

FIG. 3. Graph showing
(a) observed growth of rounding errors δ_n and $(1-\delta_n)$
(b) theoretical growth of rounding errors
(c) ratio of actual error to theoretical error
for the 10th order ACORN generator implemented in single precision real arithmetic

TABLE I. Lower and upper bounds for n_D and exact values of n_D calculated for a series of values of m and M.

m	$M = 2^{10}$			$M = 2^{20}$			$M = 2^{30}$		
	$\underline{n}_D$	n_D	$\overline{n}_D$	$\underline{n}_D$	n_D	$\overline{n}_D$	$\underline{n}_D$	n_D	$\overline{n}_D$
5	7.07	8	10.07	34.29	35	37.29	143.15	144	146.15
6	5.97	7	9.47	24.38	25	27.88	82.85	83	86.35
7	5.24	6	9.24	19.18	20	23.18	56.71	57	60.71
8	4.71	6	9.21	16.03	17	20.53	42.94	43	47.44
9	4.29	5	9.29	13.92	15	18.92	34.70	35	39.70
10	3.95	5	9.45	12.40	13	17.90	29.30	30	34.80
15	2.74	4	10.74	8.46	10	16.46	17.53	18	25.53
20	1.84	4	12.34	6.54	8	17.04	13.19	14	23.69
25	1.06	4	14.06	5.23	7	18.23	10.74	12	23.74
30	.33	4	15.83	4.18	7	19.68	9.04	11	24.54