INTERNATIONAL STANDARD

ISO 24153

First edition
2009-12-01

Random sampling and randomization procedures

Modes opératoires d'échantillonnage et de répartition aléatoires

Reference number
ISO 24153:2009(E)

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.

COPYRIGHT PROTECTED DOCUMENT

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Contents

Page

Foreword iv
Introduction v
1 Scope 1
2 Normative references 1
3 Terms, definitions, and symbols 1
4 General 5
5 Random sampling — Mechanical device methods 6
6 Pseudo-independent random sampling — Table method 7
7 Pseudo-independent random sampling — Computer method 7
8 Applications to common sampling situations 11
Annex A (normative) **Random number tables** 18
Annex B (informative) **Random number generation algorithm computer code** 22
Annex C (informative) **Random sampling and randomization computer code** 25
Bibliography 31

Foreword

ISO (the International Organization for Standardization) is a worldwide federation of national standards bodies (ISO member bodies). The work of preparing International Standards is normally carried out through ISO technical committees. Each member body interested in a subject for which a technical committee has been established has the right to be represented on that committee. International organizations, governmental and non-governmental, in liaison with ISO, also take part in the work. ISO collaborates closely with the International Electrotechnical Commission (IEC) on all matters of electrotechnical standardization.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of technical committees is to prepare International Standards. Draft International Standards adopted by the technical committees are circulated to the member bodies for voting. Publication as an International Standard requires approval by at least 75 % of the member bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO shall not be held responsible for identifying any or all such patent rights.

ISO 24153 was prepared by Technical Committee ISO/TC 69, *Applications of statistical methods*, Subcommittee SC 5, *Acceptance sampling*.

Introduction

Random sampling and randomization procedures are the cornerstone to the validity of many statistical methods used in experimentation, whether for industrial quality control and improvement purposes or for designed experiments in the medical, biological, agricultural, or other scientific fields. Many statistical standards address the conduct of such experimentation. In particular, all of the following acceptance-sampling standards have been designed on the premise that random sampling is employed to select the required sampling units for lot disposition purposes:

ISO 2859 (all parts), *Sampling procedures for inspection by attributes*

ISO 3951 (all parts), *Sampling procedures for inspection by variables*

ISO 8422, *Sequential sampling plans for inspection by attributes*

ISO 8423, *Sequential sampling plans for inspection by variables for percent nonconforming (known standard deviation)*

ISO 13448 (all parts), *Acceptance sampling procedures based on the allocation of priorities principle (APP)*

ISO 14560, *Acceptance sampling procedures by attributes — Specified quality levels in nonconforming items per million*

ISO 18414, *Acceptance sampling procedures by attributes — Accept-zero sampling system based on credit principle for controlling outgoing quality*

ISO 21247, *Combined accept-zero sampling systems and process control procedures for product acceptance*

In addition, ISO 2859-3 and ISO 21247 include provisions for random sampling to be applied to determine whether a lot should be inspected or not under skip-lot sampling procedures, and to decide which units require inspection from a production process under continuous sampling plans, respectively. Consequently, it is of great importance to the valid operation of all of the above standards that sampling be effectively random in its application.

Although the principles of this International Standard are universally applicable where random sampling is required and the sampling units can be clearly defined, preferably on the basis of discrete items, there are many situations in which the material of interest does not lend itself to being quantified on a discrete-item basis, as in the case of a bulk material. In such situations, the user is advised to consult the following ISO International Standards for appropriate guidance:

ISO 11648 (all parts), *Statistical aspects of sampling from bulk materials*

Random sampling and randomization procedures

1 Scope

This International Standard defines procedures for random sampling and randomization. Several methods are provided, including approaches based on mechanical devices, tables of random numbers, and portable computer algorithms.

This International Standard is applicable whenever a regulation, contract, or other standard requires random sampling or randomization to be used. The methods are applicable to such situations as

a) acceptance sampling of discrete units presented for inspection in lots,

b) sampling for survey purposes,

c) auditing of quality management system results, and

d) selecting experimental units, allocating treatments to them, and determining evaluation order in the conduct of designed experiments.

Information is also included to facilitate auditing or other external review of random sampling or randomization results where this is required by quality management personnel or regulatory bodies.

This International Standard does not provide guidance as to the appropriate random sampling or randomization procedures to be used for any particular experimental situation or give guidance with respect to possible sampling strategy selection or sample size determination. Other ISO standards (such as those listed in the Introduction) or authoritative references should be consulted for guidance in such areas.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 3534-1, *Statistics — Vocabulary and symbols — Part 1: General statistical terms and terms used in probability*

ISO 3534-2, *Statistics — Vocabulary and symbols — Part 2: Applied statistics*

ISO 3534-3, *Statistics — Vocabulary and symbols — Part 3: Design of experiments*

ISO 80000-2, *Quantities and units — Part 2: Mathematical signs and symbols to be used in the natural sciences and technology*

3 Terms, definitions, and symbols

For the purposes of this document, the terms and definitions given in ISO 3534-1, ISO 3534-2, ISO 3534-3, and the following apply.

3.1 Terms and definitions

3.1.1
cluster
part of a **population** (3.1.6) divided into mutually exclusive groups of **sampling units** (3.1.13) related in a certain manner

[ISO 3534-2:2006, definition 1.2.28]

3.1.2
cluster sampling
sampling (3.1.12) in which a **random sample** (3.1.8) of **clusters** (3.1.1) is selected and all the **sampling units** (3.1.13) which constitute the clusters are included in the **sample** (3.1.11)

[ISO 3534-2:2006, definition 1.3.9]

3.1.3
derangement
complete permutation
permutation of elements where no element remains in its original position in the set (e.g. {3, 1, 2} is a derangement of {1, 2, 3})

3.1.4
lot
definite part of a **population** (3.1.6) constituted under essentially the same conditions as the population with respect to the **sampling** (3.1.12) purpose

NOTE The sampling purpose can, for example, be to determine lot acceptability, or to estimate the mean value of a particular characteristic.

[ISO 3534-2:2006, definition 1.2.4]

3.1.5
multistage sampling
sampling (3.1.12) in which the **sample** (3.1.11) is selected by stages, the **sampling units** (3.1.13) at each stage being sampled from the larger sampling units chosen at the previous stage

NOTE Multistage sampling is different from multiple sampling. Multiple sampling is sampling by several criteria at the same time.

[ISO 3534-2:2006, definition 1.3.10]

3.1.6
population
⟨reference⟩ totality of items under consideration

[ISO 3534-2:2006, definition 1.2.1]

3.1.7
pseudo-independent random sampling
sampling (3.1.12) where a **sample** (3.1.11) of n **sampling units** (3.1.13) is taken from a **population** (3.1.6) in accordance with a table of random numbers or a computer algorithm designed such that each of the possible combinations of n sampling units has a particular probability of being taken (see also 4.4)

3.1.8
random sample
sample (3.1.11) selected by **random sampling** (3.1.9)

[ISO 3534-2:2006, definition 1.2.25]

3.1.9
random sampling
sampling (3.1.12) where a **sample** (3.1.11) of n **sampling units** (3.1.13) is taken from a **population** (3.1.6) in such a way that each of the possible combinations of n sampling units has a particular probability of being taken

[ISO 3534-2:2006, definition 1.3.5]

3.1.10
randomization
process by which a set of items are set into a random order

NOTE If, from a **population** (3.1.6) consisting of the natural numbers 1 to n, numbers are drawn at random (i.e. in such a way that all numbers have the same chance of being drawn), one by one, successively, without replacement, until the population is exhausted, the numbers are said to be drawn "in random order".

If these n numbers have been associated in advance with n distinct units or n distinct treatments that are then re-arranged in the order in which the numbers are drawn, the order of the units or treatments is said to be randomized.

3.1.11
sample
subset of a **population** (3.1.6) made up of one or more **sampling units** (3.1.13)

[ISO 3534-2:2006, definition 1.2.17]

3.1.12
sampling
act of drawing or constituting a **sample** (3.1.11)

[ISO 3534-2:2006, definition 1.3.1]

3.1.13
sampling unit
unit
one of the individual parts into which a **population** (3.1.6) is divided

NOTE 1 A sampling unit can contain one or more items, for example, a box of matches, but one test result will be obtained for it.

NOTE 2 A sampling unit can consist of discrete items or a defined amount of bulk material.

[ISO 3534-2:2006, definition 1.2.14]

3.1.14
sampling with replacement
sampling (3.1.12) in which each **sampling unit** (3.1.13) taken and observed is returned to the **population** (3.1.6) before the next sampling unit is taken

[ISO 3534-2:2006, definition 1.3.15]

3.1.15
sampling without replacement
sampling (3.1.12) in which each **sampling unit** (3.1.13) is taken from the **population** (3.1.6) once only without being returned to the population

[ISO 3534-2:2006, definition 1.3.16]

3.1.16
seed
numerical value or set of values used to initialize a **pseudo-independent random sampling** (3.1.7) algorithm or to establish a starting point in a table of random numbers

3.1.17
simple random sample
sample (3.1.11) selected by **simple random sampling** (3.1.18)

[ISO 3534-2:2006, definition 1.2.24]

3.1.18
simple random sampling
sampling (3.1.12) where a **sample** (3.1.11) of n **sampling units** (3.1.13) is taken from a **population** (3.1.6) in such a way that all possible combinations of n sampling units have the same probability of being taken

[ISO 3534-2:2006, definition 1.3.4]

3.1.19
stratified sampling
sampling (3.1.12) such that portions of the **sample** (3.1.11) are drawn from the different **strata** (3.1.21) and each stratum is sampled with at least one **sampling unit** (3.1.13)

[ISO 3534-2:2006, definition 1.3.6]

3.1.20
stratified simple random sampling
simple random sampling (3.1.18) from each **stratum** (3.1.21)

[ISO 3534-2:2006, definition 1.3.7]

3.1.21
stratum
mutually exclusive and exhaustive sub-population considered to be more homogeneous with respect to the characteristics investigated than the total **population** (3.1.6)

[ISO 3534-2:2006, definition 1.2.29]

3.2 Symbols

For the purposes of this document, the mathematical signs and symbols given in ISO 80000-2 and the following apply.

d_i the ith (least significant) digit, or face value of a coin or die

N lot size

n sample size

n_i the size of the ith sample

U uniformly-distributed random real variable on the open range (0, 1)

x_i the ith value of the variable x

$j!$ factorial j

$\lceil z \rceil$ ceiling function of z (returns the smallest integer greater than or equal to real value z)

$\lfloor z \rfloor$ floor function of z (returns the integer portion of real value z)

4 General

4.1 Random sampling is a prerequisite to the correct application of most sampling plans in industrial use. Similarly, randomization, which uses the principles of random sampling, is indispensable in the conduct of designed experiments, as it increases the internal validity of an experiment, allowing statistical methods to be used in the interpretation of an experiment's results. The goal of random sampling is to provide a means of applying the results of probability theory to practical problems, while avoiding any form of bias. This goal is not attainable using certain other types of sampling. For example, sampling based on such concepts as personal intuition or judgment, haphazardness, or quota-achievement are inherently biased and consequently can lead to serious errors in the decision-making process, with no provision to assess risks. Equi-probable random sampling seeks to eliminate such bias by ensuring that each unit in a lot has the same probability of being selected (sampling with replacement) or, alternatively, that every possible sample of a given size from the lot has the same probability of being selected (sampling without replacement).

4.2 Under equi-probable random sampling with replacement, the probability that a specific unit in a lot of N units is selected at any given draw is always $1/N$. There are N^n possible ordered random samples of n units from N units and, for completeness, there are $(N + n - 1)! / [n! (N - 1)!]$ possible different unordered random samples of n units from N units (see the note below).

Under simple random sampling without replacement, the probability that a unit in a lot is selected at a given draw is $1/N$ for the first draw, $1/(N - 1)$ for the second draw, $1/(N - 2)$ for the third draw, and so on. If n units are randomly selected from a lot of N units without replacement, then each combination of n units has the same probability of selection as every other combination of N units taken n at a time. The number of possible different unordered random samples of n units from a lot of N units is $N! / [n! (N - n)!]$, which is the number of combinations of N units taken n at a time. It is equally noteworthy that the number of possible ordered random samples of n units taken without replacement from a lot of N units is $N! / (N - n)!$, which is equivalent to the number of possible permutations of N units taken n at a time. It should be noted that random sampling without replacement is the most common sampling strategy used in acceptance sampling applications.

NOTE Under sampling with replacement based on a sample of, say, 3 units from 5 units, the lists {1, 1, 2}, {1, 2, 1}, and {2, 1, 1} are different when order is considered (and technically referred to as multisets or bags), but the same when order is not considered.

4.3 The goal of random sampling can only be achieved by adhering to rigorous procedures that have been carefully designed to achieve the intent of the definition. Several methods are presented in this International Standard to implement this goal. The mechanical device methods, in particular, assume that the coins and dice are unbiased, having been designed such that each side has the same probability of occurring during a toss or throw, and that the manner of tossing or throwing is being performed so as not to introduce bias. Furthermore, due to numerous deficiencies in the intrinsic implementations of random sampling methods in calculators and computer operating systems, programming languages, and software (see References [9], [10], [12], and [13] for further information), this International Standard has adopted a portable, proven method for generating random samples by computer. In addition, it should be noted that all of the methods below require that each distinct unit in a lot has been associated in advance with a distinct number from 1 to N, so that the sampling units identified as a result of the random sampling method can be unambiguously obtained from the lot.

4.4 Finally, to reduce awkwardness in presentation, the adjective "pseudo-independent" will often be dropped when referring to such a random sampling procedure or method (see Reference [8]). Furthermore, the adjective “random” will be used frequently in the sense that the noun it modifies (often a number or permutation) is the output of a process that randomly generates such a number or permutation. In addition, when examples are provided, the sample sizes involved are artificially kept small with the goal of simply illustrating the concepts involved.

5 Random sampling — Mechanical device methods

5.1 Urn method

5.1.1 Place N distinctly-numbered but otherwise physically-identical objects (e.g. tickets, chips, or balls) into an urn to unambiguously represent each of the N units in the lot and thoroughly mix the objects.

5.1.2 For sampling without replacement, blindly select objects from the urn, one by one without returning them to the urn and optionally re-mixing the objects between successive draws, until the desired number n of sampling units is obtained.

NOTE This method is commonly used by lottery agencies.

5.1.3 For sampling with replacement, blindly select objects from the urn, one by one, returning each object to the urn after each draw and thoroughly re-mixing the objects between successive draws, until the desired number n of sampling units is obtained. Using this method, the same unit may occur more than once in the sample.

5.2 Coin or die method

5.2.1 Determine the number m of coins or dice (or coin tosses or die throws) required, where N is the lot size and k is the number of sides of the device being used, according to the following equation:

$$m = \lceil \log_e N / \log_e k \rceil$$

5.2.2 Where multiple coins or dice are used, clearly associate each coin or die with a specific position in the interpretation sequence of digits d_i. Where a single coin or die is used, assign the result of the first toss or throw to the most significant digit d_m, the second toss or throw to the next most significant digit d_{m-1}, and so on.

5.2.3 Toss the coins or throw the dice and record the m ordered results d_i. Translate the results to decimal integers through the following equation:

$$y = 1 + \sum_{i=1}^{m} (d_i - 1) k^{m-i}$$

5.2.4 Repeat step 5.2.3, discarding all values that exceed N and, in the case of sampling without replacement, all values that have already been selected, until the desired number n of sampling units is obtained.

EXAMPLE 1 An inspector wishes to obtain a random sample of 4 units from a lot of 20 units and has a single coin available. From step 5.2.1, it is determined that $m = 5$ coin tosses are required to obtain each random number. It is decided in advance that a head will have a face value of 1 and a tail a face value of 2. The first sequence of tosses yields the multiset {1, 2, 1, 2, 2}, which through step 5.2.3 equates to $1 + (0)2^4 + (1)2^3 + (0)2^2 + (1)2^1 + (1)2^0 = 12$. The following three sequences of tosses yield the multisets {1, 2, 2, 2, 1}, {1, 1, 2, 2, 1}, and {2, 2, 1, 2, 2}, which equate to 15, 7, and 28, respectively. Since the value 28 exceeds the lot size, it needs to be discarded and additional sequences of tosses need to be performed until one more valid number is obtained to complete the random sample.

EXAMPLE 2 A random sample of 4 units from a lot of 50 units is required and the inspector has access to several six-sided dice of different colours. From step 5.2.1, it is determined that $m = 3$ dice are required to obtain each random number. The inspector chooses a blue, a green, and a red die and ranks them from most significant to least significant in that same order. However, it is evident upon examining the equation of 5.2.3 that numbers within the valid range from 1 to 50 will only result when the first die face is either 1 or 2. Consequently, some efficiency can be obtained by mapping the higher face values of the blue die to either 1 or 2 without distorting the outcome probabilities. The inspector decides in advance for odd face values on the blue die to be treated as 1 and even face values to be treated as 2. The first roll yields the multiset {3, 3, 4}, which through step 5.2.3 equates to $1 + (2)6^2 + (2)6^1 + (3)6^0 = 88$, which is too large but when transformed to {1, 3, 4} equates to 16. Three more rolls yield {6, 1, 3}, (which transforms to {2, 1, 3}), {5, 6, 6}, (which transforms to {1, 6, 6}), and {2, 5, 5}, which equate to 39, 36, and 65, respectively. Since the value 65 exceeds the lot size, it needs to be discarded and additional throws need to be made until one more valid number is obtained to complete the random sample.

EXAMPLE 3 The same scenario as in Example 2 exists but this time the inspector observes that three dice will produce numbers ranging from 1 to $6^3 = 216$, yet the lot size is only 50. The inspector decides in advance to map all outcomes from 1 to 200 to the range from 1 to 50 and discard any outcome greater than 200 to avoid distorting the outcome probabilities. The same four rolls from the previous example are evaluated under this mapping scheme. The multisets {3, 3, 4}, {6, 1, 3}, {5, 6, 6}, and {2, 5, 5} equate to 88, 183, 180, and 65. Multiples of 50 are subtracted from these numbers until each of them is within the range from 1 to 50 (if 0 is obtained, interpret it as N), resulting in the sample values of 38, 33, 30, and 15, respectively. A sample size of 4 units has been obtained so no further throws are necessary. Note that, mathematically, this mapping process is equivalent to applying the equation $v_2 = 1 + (v_1 - 1)$ modulo N, where v_1 is the initial value and v_2 is the value mapped into the desired range.

6 Pseudo-independent random sampling — Table method

6.1 Random number tables

Two tables of random numbers are provided in Annex A. The tables each consist of 3 600 random digits from 0 to 9, arranged in 60 rows by 60 columns. Their usage is briefly described below and in more detail in Annex A.

NOTE The digits in the table are directly analogous to the face values of a 10-sided die repeatedly thrown and recorded. The number of digits m required for a sampling application corresponds to the number of dice throws.

6.2 Basic method

6.2.1 Determine the number of digits m necessary to represent the lot size N. Where the lot size is an integral power of 10, ignore the initial digit in the lot size and interpret that remaining zeroes as equal to the lot size value (e.g. if N = 1 000, interpret the value 000 as 1 000).

6.2.2 Randomly select a starting point (i.e. row and column value) in the table using a method described in A.2.2.

6.2.3 Read the resulting digit in conjunction with the $m - 1$ digits to the right as a single number and record the value. Where the digits to the right would exceed the 60th column, treat columns 1, 2, and so on as columns 61, 62, and so on, respectively.

6.2.4 Increase the row value by one, repeat step 6.2.3, and record the value. Where this row value would exceed the 60th row, treat row 1 as the 61st row and increase the column values each by m digits.

6.2.5 Repeat step 6.2.4, discarding all values that exceed N and, in the case of sampling without replacement, all values that have already been selected, until the desired number of sampling units n is obtained.

EXAMPLE An auditor wishes to select a random sample of 5 units from a lot of 200 units. A random starting point is determined by coin tosses to be row 57 and column 59 and it is decided to use Table A.1. Since N is small in comparison to the maximum value capable of being represented by 3 digits (i.e. 1 000), the auditor decides to map the results of the range from 1 to 1 000 onto the range from 1 to 200. The following five numbers result: 848, 670, 902, 034, and 518. The translated sample values become 48, 70, 102, 34, and 118.

7 Pseudo-independent random sampling — Computer method

7.1 Overview

7.1.1 This International Standard adopts a specific system of algorithms developed in References [1], [7], and [13]. The algorithms have been designed to possess the mathematical and statistical properties required for random sampling as well as to be portable with respect to implementation in different programming languages on different computer platforms and to facilitate verification and auditing of the selected sample values, which might be required for regulatory purposes. An example implementation of the key program segments is provided in Annex B using the C programming language.

7.1.2 The system of algorithms involves two major sub-systems:

a) an optional initialization algorithm that automatically generates a quasi-random seed integer based on elapsed time from a reference date; and

b) a random number generator.

7.1.3 For verification or auditing purposes, the optional initialization algorithm mentioned in 7.1.2 a) and described in 7.2 would be by-passed with a manually-entered seed value. This value needs to be within the integer range from 1 and 2 147 483 398 inclusive. A copy of this input value is saved for records purposes when required. However, in general usage for quality control and designed experiment applications, there should be infrequent need to by-pass the option of automatic random seed generation, which should be the default option in practice.

NOTE The presentations of the steps of the algorithms in this clause have been kept in a more mathematical format to aid in programming. Programming code with clause references has been included in Annex B to supplement implementation of this clause.

7.2 Initialization algorithm

7.2.1 The initialization algorithm consists of:

a) an elapsed time computation algorithm, referenced to a fixed past date and time; and

b) a random number generation algorithm based on the uniform distribution, called a random number of times based on the output of item a) above, to obtain a random seed based on the time-based input.

7.2.2 The following algorithm determines the number of seconds that has elapsed since 2000-01-01 00:00:00 to the current date and time.

a) Capture the computer system's date and time to a string variable, save a copy of the variable for records purposes, and then parse the string into its time components (i.e. year, month, day, hour, minute, and second).

b) Compute the number of fully elapsed days d_e since the reference time point, using the current date's full four-digit year y, month m_1, and day d numerical values processed as follows:

if $m_1 < 3$, then let $m_1 = m_1 + 12$ and let $y = y - 1$

$$d_e = d + \lfloor (153\ m_1 - 457) / 5 \rfloor + 365\ y + \lfloor y / 4 \rfloor - \lfloor y / 100 \rfloor + \lfloor y / 400 \rfloor - 730\ 426$$

NOTE The equation for d_e may be slightly simplified for calendar years up to and including 2099 by replacing the terms following $\lfloor y / 4 \rfloor$ by "– 730 441".

c) Compute the total number of seconds s_e elapsed since the reference date using the quantity obtained in step b) and the time of day (in 24-hour "hh:mm:ss" format) captured in the string variable in step a) in accordance with the following equation:

$$s_e = 86\ 400\ d_e + 3\ 600\ h + 60\ m_2 + s$$

where h, m_2 and s are the hours, minutes and seconds, respectively.

NOTE 1 Some programming languages have built-in functions to perform the calculation of s_e directly. Such intrinsic functions should be validated before use, to ensure the effects of leap years and daylight saving time are properly handled.

NOTE 2 In 32-bit implementations of this algorithm, the value of s_e will increase over time to the point of causing computational overflow. Care should be taken in programming to ensure its output value is always mapped on the range from 1 to 2 147 483 398 inclusive.

d) The value resulting from step c) is the initializing seed for the random seed generator and is used to obtain the final seed. A copy of this value is saved to a separate variable for records purposes when required.

e) The number of times j that the subsequent random number generator is to be called is a random integer between 1 and 100 inclusive, based on the two least significant digits of the value obtained in step c) increased by 1, which may be expressed as follows:

$$j = s_e - 100 \lfloor s_e / 100 \rfloor + 1$$

7.2.3 The random number generator for the automatic seed generation (initialization function) algorithm takes the form of the linear congruential recurrence relation:

a) $x_{i+1} = 40\,692\, x_i \bmod 2\,147\,483\,399$,

which can be implemented on computers capable of handling 32-bit integers via the following steps:

b) $k = \lfloor x_i / 52\,774 \rfloor$;

c) $x_{i+1} = 40\,692\,(x_i - 52\,774\, k) - 3\,791\, k$;

d) If $x_{i+1} < 0$, then let $x_{i+1} = x_{i+1} + 2\,147\,483\,399$.

7.2.4 Generate the seed to the random sampling algorithm by assigning the result from 7.2.2 c) to x_i and then calling the formula in 7.2.3 j times per step 7.2.2 e), replacing x_i with x_{i+1} each time until the required number of calls are made.

7.2.5 The final value of x_{i+1} resulting from step 7.2.4 is a random integer between 1 and 2 147 483 398 inclusive and serves as the initial seed to the random sampling algorithm described in 7.3 [in particular, the value y_i in step 7.3.6 b)]. A copy of this value is saved to a separate variable for records purposes when required.

7.3 Random number generation algorithm

7.3.1 The random number generation algorithm consists of

a) a shuffling array that is populated by a uniform-distribution random number generation algorithm, and

b) a combination, uniform-distribution random number generation algorithm.

7.3.2 Create a 32-element array A to serve as a means of shuffling the output of the random sampling algorithm.

7.3.3 The following random number generator is used to populate the shuffling array:

a) $x_{i+1} = 40\,014\, x_i \bmod 2\,147\,483\,563$,

which can be implemented on 32-bit computers via the following steps:

b) $k = \lfloor x_i / 53\,668 \rfloor$;

c) $x_{i+1} = 40\,014\,(x_i - 53\,668\, k) - 12\,211\, k$;

d) If $x_{i+1} < 0$, then let $x_{i+1} = x_{i+1} + 2\,147\,483\,563$.

7.3.4 Initialize the array A by assigning the result from 7.1.3 or 7.2.5 to x_i and then calling the generator given in 7.3.3 a) 40 times, replacing x_i with x_{i+1} on each call, discarding the first 8 values, and then assigning each of the remaining 32 output values of x_{i+1} to the array in reverse order (i.e. from element 32 down to element 1).

7.3.5 Set element 1 of array A (i.e. A[1]) as the initializing value k to the combination random number generation algorithm.

7.3.6 The combination random number generator for random sample generation takes the form of the following combination of linear congruential recurrence relations and array index determination steps:

a) $x_{i+1} = 40\,014\, x_i \bmod 2\,147\,483\,563$;

b) $y_{i+1} = 40\,692\, y_i \bmod 2\,147\,483\,399$;

c) $J = \lfloor 32\, k / 2\,147\,483\,563 \rfloor + 1$;

d) $k = A[J] - y_{i+1}$;

e) $A[J] = x_{i+1}$;

f) If $k < 1$, then let $k = k + 2\,147\,483\,562$.

NOTE The two random number generators above are those described in 7.2.3 and 7.3.3 (refer to those subclauses if 32-bit equivalent implementations are required).

7.3.7 The algorithm in 7.3.6 is initialized by setting x_i to the final value of x_{i+1} from 7.3.4 and setting y_i to the value referenced in 7.2.5. The values x_{i+1} and y_{i+1} serve as the subsequent values of x_i and y_i for all subsequent calls to the algorithm. A random index J to the shuffling array A is calculated using the value of k (from 7.3.5 initially), and the difference between $A[J]$ and y_{i+1} is assigned to k, while $A[J]$ is updated with x_{i+1}. Finally, the value of k is altered if necessary to produce a positive value.

7.3.8 The output of the random sampling algorithm is the value k, which is a random number between 1 and 2 147 483 562 inclusive, scaled as a standard uniformly-distributed real variable U over the range from 0 to 1, exclusive of the endpoint values of this range, as follows: $U = k / 2\,147\,483\,563$.

7.3.9 The output from 7.3.8 may be scaled as a uniformly-distributed integer variable L over the range from 1 to N, inclusive, as follows: $L = \lfloor N\, U \rfloor + 1$.

7.3.10 To generate a random sample, steps 7.3.6 to 7.3.9 are repeated until the desired number of random values is obtained.

7.4 Audit records

When records are required to be maintained for audit purposes by a responsible authority or regulatory body, record the lot size and the sample size.

In addition, with respect to the algorithms, record the manually entered seed per 7.1.3, or if the random seed generator is used, record the

a) computer system's date and time used to compute this initial seed,

b) initial seed's value per 7.2.2 d), and

c) final seed's value per 7.2.5.

8 Applications to common sampling situations

8.1 General

8.1.1 This clause provides algorithms for several random sampling strategies to suit various practical situations.

8.1.2 Throughout this clause, U is defined as a random real variable, uniformly-distributed in the range from 0 to 1, exclusive of the endpoint values of the range, such as provided by the algorithm in 7.3. If another source is used for U and the output is known to include 1 but not 0 as the endpoint values of the range, set U equal to $1 - U$. If the alternate source of U includes 0 and 1 as endpoint values of its range, the value 1 needs to be trapped and discarded.

8.2 Random integer in a range

A random integer K in the range from M to N inclusive may be generated according to the following algorithm.

a) Generate a random real value U.

b) Set K equal to $M + \lfloor U\ (N - M + 1) \rfloor$.

8.3 Random permutation

For an array A with N distinct elements, a random permutation of N units taken n at a time may be generated according to the following shuffling algorithm.

a) Assign the N distinct element index values in original order to $A[1{:}N]$.

b) Set J equal to 1.

c) Generate a random integer K in the range from J to N inclusive.

d) Swap $A[J]$ and $A[K]$.

e) Increment J by 1.

f) If J is less than or equal to n, go to step c).

g) Obtain the random permutation from the first n values of array A.

8.4 Random derangement

For an array A with N distinct elements, a random derangement of N units may be generated according to the following algorithm.

a) Assign the N distinct element index values in original order to $A[1{:}N]$ and make a copy in array $B[1{:}N]$.

b) Using array B, generate a random permutation of N units taken N (i.e. all) at a time, using the method given in 8.3.

c) Compare the elements from 1 to N of arrays A and B for equality.

d) If any element of array B is equal to its counterpart in array A, cease the comparison and go to step b).

e) Obtain the random derangement from array B.

NOTE This algorithm can be made more efficient in steps b) and c) by comparing element $A[J]$ with element $B[J]$ as soon as $B[J]$ has been determined, rather than waiting for the full permutation of array B.

8.5 Random sampling with replacement

A single random sample of n units from a lot of N units may be generated with replacement according to the following algorithm.

a) Generate a random integer K in the range from 1 to N inclusive.

b) Repeat step a) until n values of K are obtained.

NOTE This method may be applied repeatedly to obtain any number of samples, of any size. If the resulting values of a single sample are not sorted, that sample may be used for sequential sampling inspection.

8.6 Random sampling without replacement

A single random sample of n distinct units from a lot of N units may be generated without replacement by either of the following methods.

a) Method 1

   1) Generate a random integer K in the range from 1 to N inclusive.

   2) Verify that the value of K has not been previously generated; if it is distinct, store the value, otherwise discard it.

   3) Repeat steps 1) and 2) until n different values of K are obtained.

b) Method 2

   1) Generate a random permutation of N units taken n at a time in accordance with 8.3.

   2) Use the first n values in the output array A as the random sample.

NOTE Either of these methods may be used to obtain any number of samples of any size (for such purposes as double or multiple sampling) by using the total n_t of the individual sample sizes n_i as the input value of n to the algorithm, leaving the values in original output order, then taking the first n_1 resulting values as the first sample, the next n_2 resulting values as the second sample, and so forth. Furthermore, if the resulting values of a single sample are not sorted, that sample may be used for sequential sampling inspection by inspecting each unit in the order selected.

8.7 Random sampling for continuous sampling plans (CSP)

A CSP-1 continuous sampling plan is designed for application to the quality control of a production line and alternates between qualifying periods of 100 % inspection requiring i consecutively-accepted units before being followed by periods of sampling inspection at probability f, with reversion to 100 % inspection upon finding an unacceptable unit. During periods of sampling inspection, units from the production line may be selected for inspection in accordance with either of the following methods.

a) Method 1

   1) For each unit of production, generate a random real value U.

   2) If U is less than or equal to f, choose the unit for sample inspection.

   3) Repeat steps 1) and 2) until an unacceptable unit is obtained.

b) Method 2

   1) For each production segment of n units, where n equals $1/f$, generate a random integer K over the range from 1 to n inclusive.

2) Choose the unit corresponding to K as a sampling unit for inspection.

3) Repeat steps 1) and 2) until an unacceptable unit is obtained.

NOTE For CSP-1 plans, the value f is specified as the reciprocal of an integer.

8.8 Stratified random sampling

For a lot composed of two or more strata of size N_i, select a single random sample of size n_i from each stratum i using the methods given in 8.3 or 8.6 when sampling without replacement is required, or the method given in 8.5 when sampling with replacement is required.

8.9 Single random sampling from an initially unknown lot size

A single random sample of n different units from a lot of initially unknown size, but at least equal to size n, may be obtained according to the following method (adapted from Reference [11]).

a) Assign the first n units from the lot to the sample array A[1:n].

b) If another unit exists in the lot listing, set N equal to the count of the next unit; otherwise, go to step f).

c) Generate a random integer K in the range from 1 to N inclusive.

d) If K is less than or equal to n, set A[K] equal to N.

e) Go to step b),

f) Obtain the random sample from array A and the lot size from the value N.

NOTE This method may also be used if the lot size is known.

8.10 Ordered single random sampling without replacement

A single random sample of n distinct units from a lot of N units may be generated directly in ascending order with either of the following methods.

a) Method 1 (adapted from Reference [2])

1) Initialize the following variables:

i) create array A[1:n];

ii) set L equal to N, K equal to $N - n$, and J equal to 0.

2) Increment J by 1.

3) If J is greater than n, go to step 8).

4) Generate a random real value U and set P equal to 1.

5) Set P equal to $P\ K$ / N.

6) If P is less than or equal to U:

i) set A[J] equal to $N - L + 1$ then decrement L by 1;

ii) go to step 2).

7) If P is greater than U:

   i) decrement L by 1 and K by 1;

   ii) go to step 5).

8) Obtain the random sample in ascending order from array A.

b) Method 2 (adapted from Reference [3])

1) Let C(a, b) be a function yielding the number of combinations of a units taken b at a time (also known as the binomial coefficient and equal to $a!$ / [$(a - b)!$ $b!$]).

2) Generate a random integer L in the range from 1 to C(N, n) inclusive.

3) Create array A[1:n].

4) Set K equal to 0, J equal to 1, and m equal to $n - 1$.

5) Set A[J] equal to 0.

6) If J is not equal to 1, set A[J] equal to A[$J - 1$].

7) Set A[J] equal to A[J] + 1.

8) Set R equal to C($N - A$[J], $n - J$).

9) Increment K by R.

10) If K is less than L, go to step 7).

11) Decrement K by R.

12) Increment J by 1.

13) If J is less than or equal to N, go to step 5).

14) Set A[n] equal to A[m] + $L - K$.

15) Obtain the random sample in ascending order from array A.

NOTE Due to limitations in computer representation of large integers as well as limitations in the resolution of random number generators, care needs to be taken to ensure that Method 2 is computationally feasible and not unduly biased by the random number generator being used.

EXAMPLE A random sample of 5 units in sorted order is required from a lot of 25 units. There are 25! / (20! 5!) = 53 130 possible combinations of 5 units from 25 units, which is computationally feasible using modern computers. In addition, the random number generator described in Clause 7 is selected for this sampling purpose; its maximum output value of 2 147 483 562 is slightly over 40 419 times greater than the range required so the resulting bias in the method is insignificant for practical purposes. A single random integer from 1 to 53 130 inclusive is generated as 7 319. The resulting sample set is obtained as {1, 7, 13, 18, 19}.

8.11 Cluster sampling

For a population or lot composed of clusters of related units, itemize the clusters in a list and select a random sample from this list using the methods given in 8.3 or 8.6 when sampling without replacement is required, or the method given in 8.5 when sampling with replacement is required. The resulting sample consists of the total number of units in the selected clusters.

8.12 Random sampling with probability proportional to size

For a population composed of units of different integral sizes, a random sample of units selected proportional to size may be obtained by using either of the following methods.

a) Method 1

   1) On the list of N units with varying sizes, record the cumulative sizes S_i of the units beside each successive unit.

   2) Generate a random integer K in the range from 1 to S_N inclusive, where S_N is the cumulative size of the total population.

   3) From the list, select the unit associated with the largest cumulative size not exceeding K as a sample member.

   4) Repeat steps 2) and 3) until the desired number n of sampling units (with or without replacement) is obtained.

b) Method 2

   1) From the list of N units with varying sizes, determine the maximum unit size M.

   2) Generate a pair of random integers (K, L), with K in the range from 1 to N inclusive and L in the range from 1 to M inclusive.

   3) If the size of unit K does not exceed M, choose unit K as a member of the sample.

   4) Repeat steps 2) and 3) until the desired number n of sampling units (with or without replacement) is obtained.

EXAMPLE A marketing company wishes to conduct a survey of households with a selection proportional to household size (i.e. number of occupants). A list with 10 households ranked in order of size is obtained and the sizes are: {2, 2, 3, 3, 3, 4, 4, 5, 6, 7}. The cumulative sizes for this list are: {2, 4, 7, 10, 13, 17, 21, 26, 32, 39}. A random sample of 4 households without replacement is required. Random integers between 1 and 39 inclusive are generated, yielding {7, 33, 2, 11}. The corresponding sampling units are the households with listed ranks: {3, 10, 1, 5}.

8.13 Multi-stage sampling

For a population or lot that is structured in a logical hierarchy of successively smaller groupings of units, select a random sample from the largest groupings, then subsample smaller groupings from each previously selected grouping, continuing with this procedure until the individual unit level in the hierarchy is reached. At each stage, use the random sampling methods given in 8.3 or 8.6 when sampling without replacement is required, or the method given in 8.5 when sampling with replacement is required. The number of units in the sample is the product of the number of samples taken at each stage.

EXAMPLE A lot is composed of 20 pallets with 20 boxes per pallet. Each box contains 10 units. The purchaser wishes to examine the product using a multi-stage sampling strategy. A random sample of 4 pallets is selected. From each selected pallet, a random sample of 4 boxes is then selected. Finally, from each selected box, a random sample of 3 units is selected. This procedure yields a sample of 48 units from the 4 000 units in the lot.

8.14 Randomization in designed experiments

In designed experiment applications, randomization is used to perform such activities as allocating experimental treatments to units or subjects and establishing the order of evaluating the units, including the evaluation orders for replicated designs. Either of the following randomization methods may be used.

a) Method 1

1) Assign a distinct integer from 1 to N to each element of the list of the N treatments or units, as the case may be.

2) Generate a random permutation of the N integers taken N (i.e. all) at a time.

3) Conduct the experimental activity according to the sequence resulting from step 2).

b) Method 2

1) Generate N random real variables U_i, and assign each value in the order generated to each successive element of the list of treatments or units, as the case may be.

2) Sort the list of treatments or units in ascending order according to their respective values of U.

3) Conduct the experimental activity according to the sequence resulting from step 2).

EXAMPLE 1 A medical researcher wants to test the effect of a new drug in comparison to the normally used treatment for a particular medical condition. Twelve subjects volunteer and are assigned the numbers from 1 to 12 as they join up for participation in the clinical trial. The researcher plans to allocate treatment A (new drug) to 6 subjects and treatment B (current drug) to the other 6 subjects. As a further step to reduce bias, the experimenter decides to first randomize the order of the 12 treatments before allocating them to the randomized list of 12 subjects. The index numbers of the treatments and subjects are each separately randomized using method 1. The resulting randomized list of the treatments is {B, B, A, B, A, A, B, A, A, B, B, A} and the randomized list of subjects is {3, 7, 12, 5, 1, 9, 11, 4, 10, 2, 8, 6}. The treatments may now be directly allocated to the subjects based on the order in the lists, yielding {B3, B7, A12, B5, A1, A9, B11, A4, A10, B2, B8, A6}. Consequently, treatment A is allocated to subjects {1, 4, 6, 9, 10, 12} and treatment B is allocated to subjects {2, 3, 5, 7, 8, 11}.

EXAMPLE 2 An experimenter wishes to conduct a replicated experiment, testing each of 5 units in random order at three different times. The set {1, 2, 3, 4, 5} is randomly permuted by method 1, yielding the following three ordered lists: {2, 1, 5, 4, 3}, {1, 5, 2, 3, 4}, and {4, 3, 5, 2, 1}. The three lists are assigned in order to test times 1, 2, and 3, respectively, and the experimenter tests the units according to the order specified in the list associated with each test time.

8.15 Random Latin square

A Latin square of order n is an $n \times n$ array containing symbols from some alphabet of size n, arranged so that each symbol appears exactly once in each row and exactly once in each column. It is useful in the planning of some types of designed experiments. A random Latin square of order n may be generated by the following method (from Reference [4]).

a) Create arrays $A[1{:}n, 1{:}n]$ and $C[1{:}n]$.

b) Set R equal to 1.

c) Assign integers 1 to n to array $C[1{:}n]$.

d) Set J equal to N.

e) Set C equal to 1.

f) Set I equal to 0.

g) Generate a random integer X in the range from 1 to J inclusive.

h) Set H equal to 1.

i) If I is greater than 50, go to step c).

j) If $A[H, C]$ is equal to $C[X]$, increment I by 1 and go to step g).

k) Increment H by 1.

l) If H is less than or equal to $R - 1$, go to step i).

m) Set $A[R, C]$ equal to $C[X]$ and decrement J by 1.

n) If X is greater than J, go to step r).

o) Set K equal to X.

p) Set $C[K]$ equal to $C[K + 1]$.

q) Increment K by 1; if K is less than or equal to J, go to step p).

r) Increment C by 1; if C is less than or equal to n, go to step f).

s) Increment R by 1; if R is less than or equal to n, go to step c).

t) Obtain the random Latin square from array A.

NOTE An algorithm for generating uniformly distributed random Latin squares may be found in Reference [5]. In addition, it should be noted that there is a connection between a random Latin square and the generation of random derangements of a permutation as each row and column of a Latin square is a derangement of all of the previous rows and columns.

Annex A
(normative)

Random number tables

A.1 Description

This annex provides two tables of random numbers that are applicable when access to a computerized implementation of the random number generation algorithms of this standard is not available. Each table consists of 3 600 random digits from 0 to 9, which each occur with equal frequency. The tables are each arranged in a 60-row by 60-column format to facilitate the use of the time of day to establish as a starting point. The tables were generated using the algorithms described in Clause 7.

A.2 Use

A.2.1 Number of digits and interpretation

A.2.1.1 Determine the number of digits m necessary to represent the lot size N. The number of digits is equal to the number of digits in the lot size except where the lot size is an integral power of 10; in this case, ignore the initial digit in the lot size and interpret the remaining zeroes as the number of digits required, as well as being equal to the lot size value (e.g. if $N = 100$, interpret the value 00 as 100).

A.2.1.2 When the lot size is less than or equal to one-half of 10^m, tabular entries may be interpreted based on a mapping of the observed value to the range from 1 to N, provided that bias is not introduced in the process. This may be accomplished by discarding all values that exceed kN, where $k = \lfloor 10^m / N \rfloor$, before mapping the value according to the equation $v_2 = 1 + (v_1 - 1)$ modulo N, where v_1 is the initial value and v_2 is the value mapped into the desired range.

A.2.2 Starting point(s)

A.2.2.1 Before numbers can be obtained from the tables, a strategy for choosing a starting point needs to be decided upon. The tables were designed to readily permit the time of day from a clock or watch capable of displaying time to a resolution of a second to be used for this purpose. The following is a possible method.

a) Record the current time in “hh:mm:ss” format.

b) Use the value of the seconds to determine the row value, interpreting 00 as 60.

c) Use the value of the minutes to determine the column value, interpreting 00 as 60.

d) Use the value of the hours to determine whether Table A.1 or Table A.2 is to be used, based on whether the value is odd or even.

EXAMPLE An experimenter wishes to choose a starting point in the tables for selecting a random sample from a lot of 100 units. The current time is recorded as 10:35:13. The entry coordinates to the tables are therefore row 13 and column 35 of Table A.2 (since 10 is even). The digit at that location is 6 but since two digits are needed to select a sample from 100 units, the value from column 36 would be included as well to identify the first sampling unit as 66.

A.2.2.2 Any other method that provides a random source of uniformly distributed integers over the range from 1 to 60 inclusive such as the coin or die method in 5.2 or the computer algorithm in Clause 7 (which may be used to generate a long list of random entry coordinates to be used on successive occasions) may also be used. In addition, at the conclusion of selecting a sample from the table on a particular occasion, the coordinates of the immediately following entry may be recorded and then used as the starting point for the next occasion of sampling.

EXAMPLE Continuing with the example of A.2.2.1, assume that a random sample with replacement of 10 units is required and that the direction to be used to select these numbers is downwards. The resulting sample is {66, 13, 10, 45, 32, 22, 41, 49, 22, 99}. The coordinates of the immediately following value are row 23 and column 35. These values can be recorded and used as an entry point the next time a sample is required.

A.2.2.3 The entry coordinates may also be determined by establishing an initial digit based on the row and column values from A.2.2.1 or A.2.2.2, and additional digits based on the current row and additional column values based on numbers randomly generated without replacement over the range from 1 to 60 inclusive for the remaining $m - 1$ digits required. The resulting multi-digit number would be formed in the order in which the column entries were generated.

EXAMPLE A random sample from a lot of 1 000 is required. The initial row and column are determined as 5 and 11, respectively, in Table A.1. Two more digits are required and these have been externally generated as 1 and 30, yielding the following coordinates for the first, second, and third digits, respectively: (5, 11), (5, 1), and (5, 30). The direction taken in the table is decided to be downwards. The first number is therefore 511, followed by 943, 419, 413, 899, 209, etc.

A.2.3 Dealing with table boundaries

A.2.3.1 When reading a number composed of m digits and the digits to the right of the first digit would exceed the 60th column, treat columns 1, 2, and so on as columns 61, 62, and so on, respectively. This rule may be applied within the existing table or by treating the other table of random numbers as an extension of the first table.

A.2.3.2 The usual rule is to obtain subsequent random numbers by increasing the row value by one and reading the m digits of the number based on the predetermined column values and their order. Where this row value would exceed the 60th row, treat row 1 as row 61 and increase the column values by m in the case of consecutively-used columns, or by one in the case where A.2.2.3 was used to establish the columns, and continue obtaining numbers. This rule may be applied within the existing table or by treating the other table of random numbers as an extension of the first table, providing that it is not already in use as per A.2.3.1.

A.2.4 Audit records

When records are required to be maintained for audit purposes by a responsible authority or regulatory body, record the lot size and sample size.

In addition, with respect to the tables and their use, record the

a) initial row,

b) initial column(s) and their sequence order,

c) direction taken in the table,

d) initial table used and the manner in which it was extended by the other table, if applicable, and

e) mapping used.

Table A.1 — Random numbers (for odd values)

Row i	Column j											
	5	10	15	20	25	30	35	40	45	50	55	60
	95183	14683	96585	84761	65044	65183	55567	28734	19802	56410	79127	02879
	08509	97009	47525	88791	93751	70490	17749	32927	65085	94970	55541	89466
	45448	66819	86936	95349	08657	75106	97487	85268	59208	43206	14898	29083
	02230	00022	46390	76658	91934	64676	42429	96812	30560	99913	72809	66736
5	13275	96798	51425	67147	15216	71831	16229	25862	22090	91420	24352	03550
	44439	33385	95151	92374	14683	00323	57667	78341	09004	80139	81182	87552
	17629	80967	42144	58190	24550	62189	94525	44967	15860	85739	93323	87043
	14328	77127	40397	78105	75031	99553	84296	01482	25738	32761	85035	68873
	96896	02466	86706	09507	66840	68509	38033	90785	75831	98886	00905	48343
10	09725	80938	27971	01243	29232	28799	88456	99618	20071	79865	63584	69087
	55021	37184	69480	56317	19944	56756	37514	86439	69831	15172	81398	69574
	06492	95014	54908	21591	13771	35967	78637	29918	47923	61404	63378	72394
	20604	54145	27781	35157	50127	61025	57344	36615	07766	83959	34546	67011
	20202	58870	67569	71756	76284	30909	87763	21951	67756	82597	15210	04291
15	27160	01595	64831	07126	25821	81524	12585	76273	36256	41879	33287	84361
	95089	78572	87167	65888	93358	23879	84496	16147	31130	96978	80361	85195
	74825	21529	24660	33314	64512	80550	51712	23057	53841	32470	36790	60455
	80338	94074	65731	39470	03807	72355	40407	86049	81583	06786	16673	06017
	16596	43179	42026	94264	28301	29514	60657	21732	21548	28693	15241	68944
20	34134	42056	40153	00994	14179	44447	99399	86963	71862	01306	15489	00515
	01118	98623	33695	49221	97197	21424	91691	09365	62483	98893	22106	45399
	67371	71659	30505	71239	56944	35898	02207	93274	40142	98319	41218	43739
	03485	55173	68477	12348	76971	64800	86498	42059	08942	32931	73896	27772
	33328	74045	25331	37635	39081	28786	20843	32565	24316	17888	47626	69199
25	84302	10060	25334	84920	30270	09722	61706	52863	03417	95658	74490	00143
	94775	52191	94552	99265	55079	64517	16803	13037	50984	14886	04385	67907
	51700	63604	96771	34444	30002	67975	93167	16746	97842	25589	12568	81785
	75920	13260	44283	27735	31134	97100	36706	24404	56970	44575	68832	42374
	32385	28423	46784	59222	17776	57726	56449	32109	11825	57995	91217	12802
30	13424	00587	12231	44543	62984	58391	22054	16134	73790	59050	24893	62342
	90896	00608	31377	53338	84813	76825	92192	24937	81481	01866	22641	21817
	60682	38700	34039	93512	38596	40004	71447	97193	52407	44146	77116	99965
	38746	34667	84499	70915	91391	25660	12328	35273	08135	04799	14489	19984
	51658	18422	05732	91001	98070	13591	44468	88460	66964	24038	93987	66335
35	49174	12449	97583	85835	82313	96349	92721	64617	06030	22312	94263	80291
	44215	21953	21844	44114	93162	51028	29551	66121	63959	97789	44259	90865
	33877	94654	10025	84935	94630	49660	23473	89644	67212	75851	83767	45647
	83411	43288	47832	40488	89085	69731	00790	60182	71358	22571	94204	64211
	27135	82404	52031	44648	97600	72166	70830	27701	01755	00523	01837	31304
40	29475	31431	46863	70098	98659	72035	21538	12923	76963	78288	59083	18839
	56886	38711	66126	16504	87900	74055	46028	84821	83323	35962	27522	87875
	55061	35916	15955	28228	36994	73167	17137	36572	48592	60721	97714	61215
	01646	62126	37253	24997	53016	96515	40536	39311	64151	93960	24053	87645
	41789	28167	90577	84499	25059	90583	09422	87357	55416	81135	41286	92320
45	26066	80119	44259	94514	21211	44302	29023	28138	03693	50650	38450	61118
	63559	20927	12881	25582	07872	28073	59006	55666	68690	59772	25162	87924
	35054	84077	02504	10800	75293	86466	92406	56289	79807	55271	73177	70568
	32826	26937	75563	14290	30078	70820	58639	64900	61699	34974	11738	64065
	07860	90064	91220	46786	45994	47375	70140	35592	05990	58470	82014	05265
50	79276	72512	19525	27397	88975	77137	40032	06205	06997	53504	07760	62546
	41093	04332	68677	27073	94104	58532	53616	32156	66153	00264	36374	15230
	27382	27938	91695	64013	46719	61629	33668	32391	35411	68209	33885	64050
	38984	47230	59448	97802	37987	22733	52199	12325	18625	01271	84870	10911
	51411	44221	93363	48654	42656	99464	08481	98128	66677	89441	66019	75095
55	80310	18848	23722	30788	27435	03780	85737	05561	57203	07316	98597	73621
	40082	39571	89790	65382	01447	15984	60854	72833	87320	95245	40678	89785
	86922	84354	81939	32180	32891	52704	84659	95442	86204	44040	51613	15984
	07932	69932	18796	87070	82202	05372	93506	60697	48535	89027	45719	51567
	20383	07288	50265	48321	63056	35861	80864	86357	51567	68151	11723	06990
60	45471	31340	30187	23899	36361	96780	55823	37743	06957	77884	78061	36603

Table A.2 — Random numbers (for even values)

Row i	Column j											
	5	10	15	20	25	30	35	40	45	50	55	60
	51326	91644	88971	00664	10776	90888	97107	00930	87438	23714	33246	72109
	96821	43647	41707	91062	20037	47660	34652	82087	29652	51614	11015	66187
	67104	67934	95662	85259	08546	77032	28958	26815	66683	50539	65189	22993
	93315	93110	01022	16079	74364	97582	34309	18699	24437	12839	48005	46478
5	45823	44862	86472	77354	95916	36599	52350	89866	98532	69704	33735	13696
	32573	58513	70797	53560	23115	38325	74380	99917	21721	00323	29402	27080
	74553	41700	25357	17428	66708	83630	42360	16842	40782	21345	88668	95845
	06794	48960	75160	18552	00424	25976	69852	35837	69810	64014	03045	53378
	75606	34848	74458	71100	01512	89662	01391	49140	18048	59282	77344	72419
10	97535	55169	47044	81940	16507	12220	16375	65306	38691	75019	55186	47269
	89809	48811	09494	06576	58258	73155	46712	11106	38382	70893	39503	42937
	88095	48475	07399	02165	98038	97247	48441	93066	20276	72169	49706	56553
	48854	75803	34089	45335	13056	44509	77886	65036	42619	95564	66315	49084
	97951	73015	36585	89007	12069	96858	17241	37189	44193	28270	81624	75256
15	48953	47945	81153	10121	69935	60894	23151	00113	40597	71164	61331	10930
	83334	44992	30210	99370	60425	87391	63244	54828	12155	64082	43773	66172
	31105	74546	74875	51747	41021	22338	49403	20173	75454	62122	37227	35790
	94245	19396	03648	28548	42987	78668	09292	23073	21116	07199	21398	18770
	19221	70541	58195	21383	57877	27812	06504	18289	18606	17680	09218	58984
20	33503	45776	88514	10094	28589	47548	64714	96174	10026	87111	29333	77885
	03602	99374	41918	43875	11258	83646	46042	26986	11003	94756	89972	00805
	43277	65791	12217	23767	09833	66504	82359	95754	40249	71472	61588	04428
	31966	42142	71410	28139	60147	68496	89021	01615	36565	85598	18048	32584
	19193	31952	87947	89521	65225	97987	14794	90695	69314	10359	27881	38183
25	98935	48606	27475	73462	66692	11151	21709	32836	92997	33682	12722	38906
	21880	10506	78478	85067	30375	82944	06660	40750	84252	20463	37184	27248
	81695	06183	08147	01241	16278	54886	36468	21315	49106	10291	16837	40481
	68883	60266	66180	05680	38799	40481	73524	73255	79950	42007	56334	54332
	59547	60741	56065	71467	07193	38784	08169	07389	64049	21355	86589	06583
30	91709	73154	86898	20234	05773	47157	72305	20819	57301	89018	74851	50560
	57195	97283	25156	59277	33608	73937	19341	17262	63955	41678	36229	54204
	03657	71909	82018	83110	21722	03455	30654	57890	18530	60458	57145	08764
	93373	88795	11353	44726	66989	24389	93445	53752	98703	55276	18391	53513
	35158	50868	45055	12180	29993	69555	69613	69358	96861	01667	47738	11964
35	38056	14298	10431	53147	76843	32128	46844	23407	62423	01712	46033	64425
	49152	05010	84942	25483	52825	17485	67614	12493	88626	39589	56044	21968
	80000	90734	70131	19986	34949	76990	48325	39323	66921	89134	21853	18973
	53597	22379	94302	15425	62185	27894	37281	38876	97902	34008	45051	05607
	09151	34061	64751	96631	50373	61603	84917	56084	57647	80898	55489	24602
40	99734	68144	63963	73011	22832	98145	31523	60195	34172	40637	60940	51237
	84547	89655	53120	95599	04602	07968	85748	74914	76227	07158	24432	22963
	18815	26665	25301	67754	88457	19913	96787	71084	14867	03077	89575	66834
	14169	38336	41192	56208	29069	87045	32135	25975	71643	74200	52556	30213
	90528	60501	73201	72999	30355	86428	39401	72077	48056	17853	24894	19838
45	99055	42696	14376	24907	06082	61789	03963	64664	09132	87218	64755	46107
	62530	10183	38149	70004	74983	02092	40704	01062	17000	61170	99026	24025
	74196	77214	89483	43933	80953	81268	46485	23647	98173	55947	96727	86378
	27293	56047	73998	19996	94427	09157	62999	88803	81272	22315	92708	07343
	94220	93209	32369	82003	82433	85790	47632	36285	68771	06006	37556	51601
50	68430	23169	58879	97812	39399	71469	40835	04924	30336	59222	06350	45656
	61949	23031	50698	85772	85990	36942	11098	06636	57547	73247	46229	52551
	57248	90383	23502	22642	80722	38164	12160	51707	22075	20624	91644	08780
	10777	53979	65288	39116	80635	49653	36903	33854	79873	67823	23256	31643
	06717	92287	42775	79274	90874	44006	27312	15909	25276	59863	75607	22277
55	09519	67689	13829	30992	44921	67375	94754	95322	25501	78486	99059	62524
	57335	48704	79426	49770	32989	22640	88230	66598	27685	29719	99930	26181
	14911	08271	21662	40886	53783	76430	41233	44057	28385	21751	51476	64387
	04837	08929	81607	33210	61894	17240	37617	56753	61251	49433	65644	63758
	14430	20139	15027	52208	16440	59911	57566	22227	60109	95260	21388	96686
60	68896	64599	91227	55882	60220	70202	73354	34776	55530	20599	45720	75145

Annex B
(informative)

Random number generation algorithm computer code

B.1 Overview

The following code is written in the C programming language (see Reference [6] for language details) and provides an implementation of the algorithms described in Clause 7. References to the relevant parts of Clause 7 are directly included as remarks within the programming code segments to aid in translating to other programming languages.

B.2 Demonstration program

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

/* Function prototypes */
int    SeedGen(void);
double U(void);

/* Global variables */
long  ij, Seed, Seed2, S2k;
char  str[20];
const long  m1=2147483563, m2=2147483399, mm1=2147483562,
      a1=40014, a2=40692, q1=53668, q2=52774, r1=12211, r2=3791;
const double ufac=4.656613057391769le-10;

/* Demonstration program */
int main(void)
{
  long i, nn, n, Seed1, *A, *ptr;
  char yn[4];
  time_t tnow;

  system("CLS");
  printf("Demonstration program for ISO 24153-1: Random sampling procedures -\n");
  printf("Part 1: Quality control and designed experiment applications (Clause 7)\n");
  printf("\nRandom number and seed algorithms used in a single sampling application\n");

  printf("Lot size:    ");
  scanf("%d", &nn);
  printf("Sample size: ");
  scanf("%d", &n);
  printf("\nManual seed (Y/N): ");
  scanf("%s", yn);
  if (yn[0] == 'Y' || yn[0] == 'y') {         /* manual seed option */
    printf("\nEnter an integer between 1 and 2147483398 inclusive: ");
    scanf("%d", &Seed);
    Seed1 = Seed;                             /* 7.1.3 save copy of seed              */
    S2k = 0;
    tnow = time(NULL);
    strftime(str, 20, "%Y-%m-%d %H:%M:%S", localtime(&tnow));
    }
  else {
    Seed = SeedGen();                         /* call automatic seed function of 7.2   */
```

```
    Seed1 = Seed;                                /* 7.2.5 save copy of seed              */
    }
  Seed2 = Seed;                                  /* RNG function seed parameters         */
  ij = -1;                                       /* RNG function initialization parameter */

  /* Create holding array for sample values */
  A = (long *) calloc(n, sizeof(long));
  if (A == NULL) {
     printf("Array allocation failed\n");
     exit(1);
  }

  /* Select a random sample (with replacement) */
  ptr = A;
  for (i = 0; i < n; i++)
    *(ptr++) = 1 + (long)floor(U() * nn);      /* 7.3.9 scaled output over (1;nn)       */

  /* 7.4 Program output with audit details */
  printf("\nLot size:        %d\n", nn);
  printf("Sample size:     %d\n", n);
  printf("Date and time:   %s\n", str);
  printf("Elapsed seconds: %d\n", S2k);
  printf("Seed:            %d\n", Seed1);
  printf("Selected sample:\n");
  for (i = 0; i < n; i++) printf("%8d", A[i]);
  printf("\n");
  system("PAUSE");
  return 0;
}
```

B.3 Automatic random seed generator function

When a manually-entered seed is required for verification or auditing purposes as mentioned in 7.1.3, this function is not called.

```
int SeedGen(void)
{
/* B.1.3 Automatic random seed generator function  */
/* 7.2.2 a) computer system date and time capture  */
  long   i, j, k;
  struct tm t, *ptr;
  time_t tnow, tref;

  /* Reference time: 2000-01-01 00:00:00 */
  t.tm_year = 2000 - 1900;
  t.tm_mon = 0; t.tm_mday = 1; t.tm_hour = 0;
  t.tm_min = 0; t.tm_sec = 0; t.tm_isdst = 0;

  tref = mktime(&t);
  tnow = time(NULL);
  ptr  = localtime(&tnow);
  strftime(str, 20, "%Y-%m-%d %H:%M:%S", ptr);

/* 7.2.2 b) number of complete days since 2000-01-01 00:00:00 */
/* 7.2.2 c) number of seconds since 2000-01-01 00:00:00       */
  S2k = (long)difftime(tnow, tref);

/* quasi-random seed generator */
  Seed = S2k;                                  /* 7.2.2 d) initial seed  */
  j = S2k - (S2k / 100) * 100 + 1;             /* 7.2.2 e) warm-up value */
  for (i = 1; i <= j; i++) {                   /* 7.2.4                  */
    k = Seed / q2;                             /* 7.2.3 b) RNG #2        */
    Seed = a2 * (Seed - k * q2) - k * r2;      /* 7.2.3 c) RNG #2        */
    if (Seed < 0) Seed += m2;                  /* 7.2.3 d) RNG #2        */
```

```
  }
  return Seed;                                /* 7.2.5 output seed      */
/* function additionally modifies global variables str[] and S2k */
}
```

B.4 Random number generation function

When the option for a manually-entered seed is chosen per 7.1.3, the parameter “Seed” (and therefore “Seed2”) is set to equal to the manually-entered value before calling this function.

```
double U(void)
{
/* B.1.4 Random number generation function */
  int j, k, i1;
  static long k1, Shuffle[32];

  if (ij < 0) {                               /* 7.3.1 a) initialize shuffling array */
    for (j = 39; j >= 0; j--) {               /* 7.3.4 fill shuffling array          */
      k = Seed / q1;                          /* 7.3.3 b) RNG #1                     */
      Seed = a1 * (Seed - k * q1) - k * r1;   /* 7.3.3 c) RNG #1                     */
      if (Seed < 0) Seed += m1;               /* 7.3.3 d) RNG #1                     */
      if (j <= 31) Shuffle[j] = Seed;         /* 7.3.4                               */
    }                                         /* final Seed value; input to 7.3.6 a) */
    ij = 0;                                   /* disable further initialization      */
    k1 = Shuffle[0];                          /* input to 7.3.6 c)                   */
  }
/* 7.3.6 combined random number generator (CRNG) */
  k = Seed / q1;                              /* 7.3.6 a) RNG #1; input from 7.3.4   */
  Seed = a1 * (Seed - k * q1) - k * r1;       /* 7.3.6 a) RNG #1                     */
  if (Seed < 0) Seed += m1;                   /* 7.3.6 a) RNG #1                     */
  k = Seed2 / q2;                             /* 7.3.6 b) RNG #2; input from 7.2.5   */
  Seed2 = a2 * (Seed2 - k * q2) - k * r2;     /* 7.3.6 b) RNG #1                     */
  if (Seed2 < 0) Seed2 += m2;                 /* 7.3.6 b) RNG #1                     */
  i1 = floor(32.0 * k1 / m1);                 /* 7.3.6 c) array index calculation    */
  k1 = Shuffle[i1] - Seed2;                   /* 7.3.6 d) unscaled temporary output  */
  Shuffle[i1] = Seed;                         /* 7.3.6 e) update shuffling array     */
  if (k1 < 1) k1 += mm1;                      /* 7.3.6 f) unscaled integer output    */
  return (k1*ufac);                           /* 7.3.8 scaled real output over (0;1) */
}
```

Annex C
(informative)

Random sampling and randomization computer code

C.1 Introduction

This annex provides C language implementations (see Reference [6] for language details) of selected algorithms described in Clause 8. The program code of this annex is written to illustrate the descriptions in Clause 8 and is not necessarily optimal in its structure.

C.2 Demonstration program

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/* Function prototypes */
void   RandPermNM(long m, long n, long A[]);
void   RandPermN(long n, long A[]);
void   RandDerang(long n, long A[], long B[]);
void   RSWOR(long n, long m, long A[]);
void   SRSWORIULS(long n, long m, long B[], long A[]);
void   OSRSWORA(long n, long m, long A[]);
void   OSRSWORB(long n, long m, long l, long A[]);
void   RLS(long n, long A[][]);
long   RandIntMN(long m, long n);
long   RandInt1N(long n);
long   C(long m, long n);
double U(void);

/* Global variables */
long  Seed;
const long ka = 16807, kq = 127773, kr = 2836, km = 2147483647;

/* Demonstration program */
int main(void)
{
  long  i, j, m, n;
  system("CLS");
  printf("Clause 8 demonstration program\n\n");

  Seed = 543210; m = 1; n = 100;
  printf("8.2 Random integer between m and n\n");
  printf("m, n, i:  %d  %d  %d\n", m, n, RandIntMN(m, n));
  system("PAUSE"); printf("\n");

  Seed = 543210; n = 100;
  printf("8.2 Random integer between 1 and n\n");
  printf("n, i:  %d  %d\n", n, RandInt1N(n));
  system("PAUSE"); printf("\n");

  Seed = 543210; m = 5; n = 10;
  long A[n];
  for (i = 0; i < n; i++) A[i] = i+1;
  printf("8.3 Random permutation of n items taken m at a time\n");
  RandPermNM(m, n, A);
  printf("m, n:  %d  %d\n", m, n);
  for (i = 0; i < m; i++) printf("%d ", A[i]);
```

```
  printf("\n");
  system("PAUSE"); printf("\n");

  Seed = 543210; n = 10;
  long A1[n];
  for (i = 0; i < n; i++) A1[i] = i+1;
  printf("8.3 Random permutation of n items taken all at a time\n");
  RandPermN(n, A1);
  printf("n:  %d\n", n);
  for (i = 0; i < n; i++) printf("%d ", A1[i]);
  printf("\n");
  system("PAUSE"); printf("\n");

  n = 10;
  long A2[n];
  printf("8.4 Random derangement\n");
  RandDerang(n, A1, A2);
  for (i = 0; i < n; i++) printf("%d ", A2[i]);
  printf("\n");
  system("PAUSE"); printf("\n");

  Seed = 543210; n = 5; m = 20;
  long A3[n];
  printf("8.6 Random sampling without replacement\n");
  RSWOR(n, m, A3);
  printf("m, n:  %d  %d\n", m, n);
  for (i = 0; i < n; i++) printf("%d ", A3[i]);
  printf("\n");
  system("PAUSE"); printf("\n");

  Seed = 543210; n = 5; m = 20;
  long A4[n], A5[m + 2];
  for (i = 0; i < m+2; i++) A5[i] = 0;
  for (i = 0; i < m; i++) A5[i] = i+1;
  printf("8.9 Single random sampling from an initially unknown lot size\n");
  SRSWORIULS(n, m, A5, A4);
  printf("m, n:  %d  %d\n", m, n);
  for (i = 0; i < n; i++) printf("%d ", A4[i]);
  printf("\n");
  system("PAUSE"); printf("\n");

  Seed = 543210; n = 5; m = 20;
  long A6[n];
  printf("8.10 Method A: ordered single random sampling without replacement\n");
  OSRSWORA(n, m, A6);
  printf("m, n:  %d  %d\n", m, n);
  for (i = 0; i < n; i++) printf("%d ", A6[i]);
  printf("\n");
  system("PAUSE"); printf("\n");

  Seed = 543210; n = 5; m = 20;
  long A7[n+1], b, d;
  b = C(m, n); d = RandInt1N(b);
  printf("8.10 Method B: ordered single random sampling without replacement\n");
  OSRSWORB(n, m, d, A7);
  printf("m, n, d, C(m,n):  %d  %d  %d  %d\n", m, n, d, b);
  for (i = 1; i <= n; i++) printf("%d ", A7[i]);
  printf("\n");
  system("PAUSE"); printf("\n");

  Seed = 543210; n = 8;
  long A8[n+1][n+1];
  printf("8.15 Random Latin square\n");
  RLS(n, A8);
  printf("Order: %d\n", n);
  for (i = 1; i <= n; i++) {
    for (j = 1; j <= n; j++) {
      printf("%3d ", A8[i][j]);
```

```
      }
    printf("\n");
  }
  printf("\n");
  system("PAUSE");

  return 0;
}
```

C.3 Functions

```
/*************************************************************************/
long RandInt1N(n)
long n;
{
/* 8.2 Random integer in a range (1 to n inclusive) */
  return (1 + (long)floor(U() * n));
}

/*************************************************************************/
long RandIntMN(m, n)
long m, n;
{
/* 8.2 Random integer in a range (m to n inclusive; m < n) */
  return (m + (long)floor(U() * (n - m + 1)));
}

/*************************************************************************/
void RandPermN (n, A)
long n, A[];
{
/* 8.3 Random permutation of N items taken all at a time */
/* A[] is both the input and output array; output in A[] */
  long j, k, temp;
  for (j = 0; j < n-1; j++) {
    k = RandIntMN(j, n-1);
    temp = A[j]; A[j] = A[k]; A[k] = temp;
    }
  return;
}

/*************************************************************************/
void RandPermNM (m, n, A)
long m, n, A[];
{
/* 8.3 Random permutation of n items taken m at a time        */
/* A[] is both the input and output array; output in A[0:m-1] */
  long j, k, temp;
  if (m == n) {m = n - 1;}
  for (j = 0; j < m; j++) {
    k = RandIntMN(j, n-1);
    temp = A[j]; A[j] = A[k]; A[k] = temp;
    }
  return;
}

/*************************************************************************/
void RandDerang (n, A1, A2)
long n, A1[], A2[];
{
/* 8.4 Random derangement                          */
/* A1[] is the input array; A2[] is the output array */
  long i, iFlag, n1;
  for (i = 0; i < n; i++)
```

```
    A2[i] = A1[i];      /* make a copy */
  iFlag = 1; n1 = n;
  for (;;) {
    RandPermN(n1, A2);
    for (i = 0; i < n1; i++) {
      if (A2[i] == A1[i]) {iFlag = 0; break;}
      }
    if (iFlag == 0) iFlag = 1;
    else break;
  }
return;
}

/*************************************************************************/
void RSWOR (n, m, A)
long n, m, A[];
{
/* 8.6 Random sampling without replacement */
/* n = sample size; m = lot size           */
/* A[] is the sample output array          */
  long i, k;
  long B[m];  /* array to keep track of units chosen */
  for (i = 0; i < m; i++) B[i] = 0;
  i = -1;
  do {
    k = RandInt1N(m);
    if (B[k] == 0) {
      B[k] = 1; i = i + 1; A[i] = k;
      }
  } while (i < n);
  return;
}

/*************************************************************************/
void SRSWORIULS (n, m, B, A)
long n, m, B[], A[];
{
/* 8.9 Single random sampling from an initially unknown lot size   */
/* n = sample size; m = lot size; (n < m)                          */
/* A[] is the sample output array                                  */
/* B[] simulates a lot of unknown size (0 indicates lot concluded) */
  long k, v;
  m = 0;        /* lot size counter */
  for (;;) {
    m = m + 1;
    v = B[m-1];
    if (v == 0) {m = m - 1; break;}
    if (m <= n) A[m-1] = v;
    else {
      k = RandInt1N(m);
      if (k <= n) A[k-1] = v;
    }
  }
  return;
}

/*************************************************************************/
void OSRSWORA (n, m, A)
long n, m, A[];
{
/* 8.10 a) Method A: ordered single random sampling without replacement */
/* n = sample size; m = lot size; (n < m)                               */
/* A[] is the sample output array                                       */
  long   j, k, k1;
  double p, x;
  k = m - n; k1 = m; j = 0;
OsrsworA1:
  j = j + 1;
```

```
  if (j > n) goto OsrsworA3;
  x = U(); p = 1.0;
OsrsworA2:
  p = p * k / k1;
  if (p <= x) {
    A[j-1] = m - k1 + 1; k1 = k1 - 1;
    goto OsrsworA1;
    }
  else {
    k1 = k1 - 1; k = k - 1;
    goto OsrsworA2;
    }
OsrsworA3:
  return;
}

/*********************************************************************/
void OSRSWORB (n, m, lx, A)
long n, m, lx, A[];
{
/* 8.10 b) Method B: ordered single random sampling without replacement */
/* Finds the combination set of m things taken n at a time              */
/* for a given lexicographical index.                                   */
/* n = sample size; m = lot size; (n < m)                               */
/* lx = lexicographical index of combination sought [1 <= lx <= C(m,n)] */
/* A[] is the sample output array                                       */
  long i, k, n1, r;
  k = 0; n1 = n - 1;
  for (i = 1; i < n; i++) {
    A[i] = 0;
    if (i != 1) A[i] = A[i-1];
OsrsworB1:
    A[i] = A[i] + 1;
    r = C(m - A[i], n - i);
    k = k + r;
    if (k < lx) goto OsrsworB1;
    k = k - r;
  }
  A[n] = A[n1] + lx - k;
  return;
}

/*********************************************************************/
void RLS (n, A)
long n, A[n+1][n+1];
{
/* 8.15 Random Latin square                          */
/* n = order; A[][] is the output random Latin square */
  long B[n+1], h, i, j, k, r, c, x;
  for (r = 1; r <= n; r++) {
Rls1:
    for (i = 1; i <= n; i++) B[i] = i;
    j = n;
    for (c = 1; c <= n; c++) {
      i = 0;
Rls2:
      x = floor(U() * j + 1);
      for (h = 1; h <= r-1; h++) {
        if (i > 50) goto Rls1;        /* row no good    */
        if (A[h][c] == B[x]) {
          i = i + 1; goto Rls2;       /* column no good */
          }
      }
      A[r][c] = B[x]; j = j - 1;
      for (k = x; k <= j; k++) B[k] = B[k + 1];
    }
  }
  return;
```

```
}

/*************************************************************************/
long C(m, n)
long m, n;
{
/* 8.10 b) Method B: auxiliary function                                  */
/* Calculates the number of combinations of m things taken n at a time. */
  long i, k, x, n1;
  n1 = n; k = m - n1;
  if (n1 < k) {k = n1; n1 = m - k;}
  x = n1 + 1;
  if (k == 0) {x = 1;}
  if (k >= 2) {
     for (i = 2; i <= k; i++)
         x = (x * (n1 + i)) / i;
  }
  return x;
}

/*************************************************************************/
double U(void)
{
/* Source: bibliographic reference [12]                */
/* RNG based on:  x[i+1] = 16807 * x[i] mod 2147483647 */
/* Included for illustration purposes only             */
  long k;
  k = Seed / kq;
  Seed = ka * (Seed - k * kq) - kr * k;
  if (Seed < 0) {Seed = Seed + km;}
  return (1.0 * Seed / km);
}
```

Bibliography

[1] BAYS, C. and DURHAM, S.D. Improving a Poor Random Number Generator. *ACM Transactions on Mathematical Software*, **2** (1), 1976, pp. 59-64

[2] BISSELL, A.F. Ordered random selection without replacement. *Applied Statistics*, **35**, 1986, pp. 73-75

[3] BUCKLES, B.P. and LYBANON, M. Algorithm 515, Generation of a Vector from the Lexicographical Index. *ACM Transactions on Mathematical Software*, **3** (2), 1977, pp. 180-182

[4] BYERS, J.A. Random selection algorithms for spatial and temporal sampling. *Computers in Biology and Medicine*, **26**, 1996, pp. 41-52

[5] JACOBSON, M.T. and MATTHEWS, P. Generating uniformly distributed random Latin squares. *Journal of Combinatorial Design*, **4**, 1996, pp. 405-437

[6] ISO/IEC 9899:1999, *Programming languages — C*

[7] L'ECUYER, P. An Efficient and Portable Combined Random Number Generator. *Communications of the ACM*, **31** (6), 1988, pp. 742-749, 774

[8] MARSAGLIA, G. Random Number Generators. *Journal of Modern Applied Statistical Methods*, **2** (1), 2003, pp. 2-13

[9] MCCULLOUGH, B.D. Assessing the Reliability of Statistical Software: Part II. *The American Statistician*, Vol. 53, No. 2 (May), 1999, pp. 149-159

[10] MCCULLOUGH, B.D. and WILSON, B. On the Accuracy of Statistical Procedures in Microsoft EXCEL 97. *Computational Statistics and Data Analysis*, **31** (1), 1999, pp. 27-37

[11] MCLEOD, A.I. and BELLHOUSE, D.R. A convenient algorithm for drawing a simple random sample. *Applied Statistics*, **32**, 1983, pp. 182-184

[12] PARK, S.K. and MILLER, K.W. Random Number Generators: Good Ones are Hard to Find. *Communications of the ACM*, **31** (10), 1988, pp. 1192-1201

[13] PRESS, W.H., Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P. *Numerical Recipes in Fortran 77: The Art of Scientific Computing*, Second Edition (Volume 1 of Fortran Numerical Recipes), Cambridge University Press, Cambridge, UK, 1992, 2001

[14] SOM, R.K. *A Manual of Sampling Techniques.* Heinemann Educational Books Ltd., London, 1973

ICS 03.120.30

Price based on 31 pages