Module 21.3: nag_rand_discrete
Random Numbers from Discrete Distributions

`nag_rand_discrete` provides procedures for generating sequences of independent pseudo-random numbers from discrete distributions.

Contents

Introduction

1 Terminology

This module is concerned with the generation of sequences of random numbers from discrete distributions.

Strictly speaking, the generated numbers are *pseudo-random* rather than true random numbers; however, their statistical properties — independence, randomness, etc. — are similar to those of true random numbers. In this module, the term 'random' will be used throughout, although strictly we mean 'pseudo-random'.

2 Discrete Distributions

This module provides procedures for the following discrete distributions: binomial, negative binomial, and hypergeometric (see Dagpunar [1], Kendall and Stuart [2], Knuth [3], Morgan [4] and Ripley [5] for further reading). The generated numbers are *integers*.

The method used consists of first setting up a *reference vector* for the required distribution and then using the reference vector to generate random integers from the distribution. Reference vectors are stored in structures of the derived type `nag_ref_vec_wp`.

Most of the procedures are *subroutines* which offer the options of *either* generating an array of random integers, *or* returning a reference vector *or* both. The procedure `nag_rand_ref_vec` uses a reference vector that has already been set up by one of the other procedures, to generate random integers from any distribution; it is a function, which may be either *scalar valued* or *array valued*.

Thus, random integers from a discrete distribution may be generated in any of the following ways (we use the procedure `nag_rand_binom` for the binomial distribution as an example):

(i) call `nag_rand_binom` to generate an array `iv` of random integers (the number of integers generated is determined by the size of `iv`):

```
call nag_rand_binom( n, p, iv=iv(1:r), seed=seed )
```

(ii) call `nag_rand_binom` to set up a reference vector `ref`; then make one or more calls to `nag_rand_ref_vec` to generate random integers:

```
call nag_rand_binom( n, p, ref=ref )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

(iii) call `nag_rand_binom` to simultaneously generate a stream of random integers *and* to set up a reference vector for subsequent use. The reference vector may then be used in one or more calls to `nag_rand_ref_vec` to generate new random integers:

```
call nag_rand_binom( n, p, iv=iv(1:r), ref=ref, seed=seed )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

3 Initialization of the Seed

All the procedures in this module make use of an argument `seed`, which is a structure of type `nag_seed_wp`. This must be initialized before use by calling the procedure `nag_rand_seed_set`. Both the type and its initialization procedure are defined by the module `nag_rand_util` (21.1), and described in its module document. However, they are also accessible via the `USE` statement for this module.

Procedure: nag_rand_binom

1 Description

nag_rand_binom is concerned with generating random integers from a binomial distribution of the number of successes, k, in n independent trials, given that each trial has probability of success p. The distribution has a probability density function (PDF) defined by

$$P(K = k) = \left(\begin{array}{c} n \\ k \end{array} \right) p^k (1-p)^{n-k} \qquad k = 0, 1, \ldots, n.$$

The procedure may be used either to generate random integers from the specified distribution (if the optional arguments seed and iv are present), or to return a reference vector (if the optional argument ref is present), or both. The reference vector may be passed to the procedure nag_rand_ref_vec to generate further random integers from the same distribution.

Note: all the output arguments of this procedure are optional. However, at least one output argument must be present in every call statement.

2 Usage

```
USE nag_rand_discrete
CALL nag_rand_binom(n, p  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

r — the number of random integers to be generated

3.1 Mandatory Arguments

n — integer, intent(in)

Input: the number of trials of the distribution.

Constraints: n > 0.

p — real(kind=*wp*), intent(in)

Input: the probability of success in each independent trial.

Constraints: $0.0 <$ p < 1.0.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

iv(r) — integer, intent(out), optional

Output: the generated random integers.

Constraints: iv must be present, if ref is absent.

seed — type(nag_seed_*wp*), intent(inout), optional

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

Constraints: `seed` must be present, if `iv` is present.

ref — type(nag_ref_vec_*wp*), intent(out), optional

Output: the reference vector, which may be passed to `nag_rand_ref_vec` in order to generate additional random integers from the same distribution.

Constraints: `ref` must be present, if `iv` is absent.

Note: to reduce the risk of corrupting the data accidentally, the components of this structure are private.
The procedure allocates approximately $20 + 20\sqrt{np(1-p)}$ real(kind=*wp*) elements of storage to the structure. If you wish to deallocate this storage when the structure is no longer required, you must call the procedure `nag_deallocate`, as illustrated in Example 1.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	Description
101	Optional argument is present but not needed. `seed` is present when `iv` is not present.

5 Examples of Usage

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. The following examples illustrate different ways to generate r random integers from a binomial distribution.

In the first example a call to this procedure generates an array `iv` of random integers (the number of integers generated is determined by the size of `iv`):

```
call nag_rand_binom( n, p, iv=iv(1:r), seed=seed )
```

The second example calls this procedure to set up a reference vector `ref`; then one or more calls to `nag_rand_ref_vec` generates random integers:

```
call nag_rand_binom( n, p, ref=ref )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

The third example calls this procedure to generate an array of random integers simultaneously, *and* to set up a reference vector for subsequent use. The reference vector may then be used in one or more calls to nag_rand_ref_vec to generate new random integers:

```
call nag_rand_binom( n, p, iv=iv(1:r), ref=ref, seed=seed )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

6 Further Comments

6.1 Algorithmic Detail

The reference vector is found by a recurrence relation if $np(1-p) < 50$; otherwise Stirling's approximation is used.

Procedure: nag_rand_neg_binom

1 Description

nag_rand_neg_binom returns random integers from a negative binomial distribution of the number of successes, k, before n failures, given that each trial has probability of success p. The distribution has a probability density function (PDF) defined by

$$P(K = k) = \left(\begin{array}{c} n+k-1 \\ k \end{array} \right) p^n (1-p)^k \qquad k = 0, 1, \ldots .$$

The procedure may be used either to generate random integers from the specified distribution (if the optional arguments seed and iv are present), or to return a reference vector (if the optional argument ref is present), or both. The reference vector may be passed to the procedure nag_rand_ref_vec to generate further random integers from the same distribution.

Note: all the output arguments of this procedure are optional. However, at least one output argument must be present in every call statement.

2 Usage

```
USE nag_rand_discrete
CALL nag_rand_neg_binom(n, p  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

r — the number of random integers to be generated

3.1 Mandatory Arguments

n — integer, intent(in)

Input: the number of failures of the distribution.

Constraints: n > 0.

p — real(kind=*wp*), intent(in)

Input: the probability of failure in each independent trial.

Constraints: $0.0 <$ p < 1.0.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

iv(r) — integer, intent(out), optional

Output: the generated random integers.

Constraints: iv must be present if ref is absent.

seed — type(nag_seed_*wp*), intent(inout), optional

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

Constraints: `seed` must be present if `iv` is present.

ref — type(nag_ref_vec_*wp*), intent(out), optional

Output: the reference vector, which may be passed to `nag_rand_ref_vec` in order to generate additional random integers from the same distribution.

Constraints: `ref` must be present if `iv` is absent.

Note: to reduce the risk of corrupting the data accidentally, the components of this structure are private.
The procedure allocates approximately $20 + (20\sqrt{np + 30p})/(1 - p)$ real(kind=*wp*) elements of storage to the structure. If you wish to deallocate this storage when the structure is no longer required, you must call the procedure `nag_deallocate`, as illustrated in Example 1.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	Description
101	Optional argument is present but not needed. `seed` is present when `iv` is not present.

5 Examples of Usage

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. The following examples illustrate different ways to generate r random integers from a negative binomial distribution.

In the first example a call to this procedure generates an array `iv` of random integers (the number of integers generated is determined by the size of `iv`):

```
call nag_rand_neg_binom( n, p, iv=iv(1:r), seed=seed )
```

The second example calls this procedure to set up a reference vector `ref`; then one or more calls to `nag_rand_ref_vec` generates random integers:

```
call nag_rand_neg_binom( n, p, ref=ref )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

The third example calls this procedure to generate an array of random integers simultaneously, *and* to set up a reference vector for subsequent use. The reference vector may then be used in one or more calls to nag_rand_ref_vec to generate new random integers:

```
call nag_rand_neg_binom( n, p, iv=iv(1:r), ref=ref, seed=seed )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

6 Further Comments

6.1 Algorithmic Detail

If $np < 50$, a recurrence relation is used to generate the reference vector; otherwise, Stirling's approximation is used.

Procedure: nag_rand_hypergeo

1 Description

nag_rand_hypergeo returns random integers from a hypergeometric distribution of the number of specified items, k, in a sample of size l, taken from a population of size n with m specified items in it. The distribution has a probability density function (PDF) defined by

$$P(K=k) = \frac{\binom{m}{k}\binom{n-m}{l-k}}{\binom{n}{l}} \qquad k = \max(0, m+l-n), \ldots, \min(l, m).$$

The procedure may be used either to generate random integers from the specified distribution (if the optional arguments seed and iv are present), or to return a reference vector (if the optional argument ref is present), or both. The reference vector may be passed to the procedure nag_rand_ref_vec to generate further random integers from the same distribution.

2 Usage

```
USE nag_rand_discrete
CALL nag_rand_hypergeo(l, m, n, iv, seed, ref  [, optional arguments])
    or
CALL nag_rand_hypergeo(l, m, n, iv, seed  [, optional arguments])
    or
CALL nag_rand_hypergeo(l, m, n, ref  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

r — the number of random integers to be generated

3.1 Mandatory Arguments

l — integer, intent(in)

Input: the sample-size parameter of the distribution.

Constraints: $0 \leq$ l $\leq$ n.

m — integer, intent(in)

Input: the number-of-specified-items parameter of the distribution.

Constraints: $0 \leq$ m $\leq$ n.

n — integer, intent(in)

Input: the population-size parameter of the distribution.

Constraints: n > 0.

iv(r) — integer, intent(out)

Output: the generated random integers.

Constraints: `iv` must be supplied if `ref` is *NOT* supplied.

seed — type(nag_seed_*wp*), intent(inout)

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

Constraints: `seed` must be supplied if `iv` is supplied.

ref — type(nag_ref_vec_*wp*), intent(out)

Output: the reference vector, which may be passed to `nag_rand_ref_vec` in order to generate additional random integers from the same distribution.

Constraints: `ref` must be supplied if `iv` is *NOT* supplied.

Note: to reduce the risk of corrupting the data accidentally, the components of this structure are private.
The procedure allocates approximately $20 + \sqrt{lm(n-m)(n-l)/n^3}$ real(kind=*wp*) elements of storage to the structure. If you wish to deallocate this storage when the structure is no longer required, you must call the procedure `nag_deallocate`, as illustrated in Example 1.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
320	The procedure was unable to allocate enough memory.

5 Examples of Usage

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. The following examples illustrate different ways to generate r random integers from a hypergeometric distribution.

In the first example a call to this procedure generates an array `iv` of random integers (the number of integers generated is determined by the size of `iv`):

```
call nag_rand_hypergeo(l, m, n, iv(1:r), seed )
```

The second example calls this procedure to set up a reference vector `ref`; then one or more calls to `nag_rand_ref_vec` generates random integers:

```
call nag_rand_hypergeo(l, m, n, ref )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

The third example calls this procedure to generate an array of random integers simultaneously, *and* to set up a reference vector for subsequent use. The reference vector may then be used in one or more calls to nag_rand_ref_vec to generate new random integers:

```
call nag_rand_hypergeo(l, m, n, iv(1:r), seed, ref )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

6 Further Comments

6.1 Algorithmic Detail

If $lm(n-l)(n-m) < 50n^3$, a recurrence relation is used to generate the reference vector; otherwise Stirling's approximation is used.

Procedure: nag_rand_user_dist

1 Description

nag_rand_user_dist returns random integers from any discrete distribution depending on the type of information contained in the vector p, which may be either a probability density function (PDF) or a cumulative distribution function (CDF). The procedure may be used either to generate random integers from the information contained in p (if the optional arguments seed and iv are present), or to return a reference vector (if the optional argument ref is present), or both. The reference vector may be passed to the procedure nag_rand_ref_vec to generate further random integers from the same distribution.

The distribution is assumed to cover m consecutive integers $a, a+1, \ldots, a+m-1$. You must supply the initial value a in the argument val, and m probabilities in the argument p.

Note: all the output arguments of this procedure are optional. However, at least one output argument must be present in every call statement.

2 Usage

```
USE nag_rand_discrete
CALL nag_rand_user_dist(p, val  [, optional arguments])
```

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '**x**(n)' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the values of the following problem parameters from the shape of the supplied arrays.

$m > 1$ — the number of data points that define the PDF or CDF
r — the number of random integers to be generated

3.1 Mandatory Arguments

p(m) — real(kind=*wp*), intent(in)

Input: the probabilities which define the discrete distribution. For a PDF, p(i) must hold the probability of the value $a+i-1$, while for a CDF p(i) must hold the probability of any of the values $a, a+1, \ldots, a+i-1$.

Constraints: for a PDF, p(i) must satisfy $0.0 \leq$ p(i) ≤ 1.0 and must not all be zero; while for a CDF p(i) $\leq 1.0 + m \times$ EPSILON(1.0_*wp*)and the p(i) must be in non-decreasing order.

val — integer, intent(in)

Input: the value a of the variate to which the probability in p(1) corresponds; that is, p(1) = Prob $(K = a)$.

3.2 Optional Arguments

Note. Optional arguments must be supplied by keyword, not by position. The order in which they are described below may differ from the order in which they occur in the argument list.

pdf — logical, intent(in), optional

Input: the type of information contained in p.
If pdf = .true., p contains a probability density function (PDF);
if pdf = .false., p contains a cumulative distribution function (CDF).

Default: pdf = .true..

iv(*r*) — integer, intent(out), optional

Output: the generated random integers.

Constraints: `iv` must be present if `ref` is absent.

seed — type(nag_seed_*wp*), intent(inout), optional

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

Constraints: `seed` must be present if `iv` is present.

ref — type(nag_ref_vec_*wp*), intent(out), optional

Output: the reference vector, which may be passed to `nag_rand_ref_vec` in order to generate additional random integers from the same distribution.

Constraints: `ref` must be present if `iv` is absent.

Note: to reduce the risk of corrupting the data accidentally, the components of this structure are private.
The procedure allocates approximately $5+1.4m$ real(kind=*wp*) elements of storage to the structure. If you wish to deallocate this storage when the structure is no longer required, you must call the procedure `nag_deallocate`, as illustrated in Example 1.

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document `nag_error_handling` (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to `nag_set_error` before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.
305	Invalid absence of an optional argument.
320	The procedure was unable to allocate enough memory.

Warnings (error%level = 1):

error%code	Description
101	Optional argument is present but not needed. `seed` is present when `iv` is not present.
102	a CDF is supplied in `p(`i`)`, but `p(`m`)` $> 1.0+m\times$`EPSILON(1.0_`*wp*`)`. This may be due to rounding errors. The procedure has scaled the probabilities to 1.0 to ensure that the reference vector is set up correctly.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

Assume that all relevant arguments have been declared correctly as described in Section 3, and that input and input/output arguments have been appropriately initialized. The following examples illustrate different ways to generate r random integers from a discrete distribution whose PDF is only known.

In the first example a call to this procedure generates an array `iv` of random integers (the number of integers generated is determined by the size of `iv`):

```
call nag_rand_user_dist( p, val, iv=iv(1:r), seed=seed )
```

The second example calls this procedure to set up a reference vector `ref`; then one or more calls to `nag_rand_ref_vec` generate random integers:

```
call nag_rand_user_dist( p, val, ref=ref )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

The third example calls this procedure to generate an array of random integers simultaneously, *and* to set up a reference vector for subsequent use. The reference vector may then be used in one or more calls to `nag_rand_ref_vec` to generate new random integers:

```
call nag_rand_user_dist( p, val, iv=iv(1:r), seed=seed, ref=ref )
 . . .
iv(1:r) = nag_rand_ref_vec( seed, ref, r )
 . . .
```

Procedure: nag_rand_ref_vec

1 Description

nag_rand_ref_vec returns random integers from a reference vector that is set up by any discrete distribution or a distribution that is specified in terms of the PDF (probability density function) or CDF (cumulative density function). It is a generic function: the result may be scalar or array valued. It may be used to generate further random integers from the distribution that supplies the reference vector.

2 Usage

```
USE nag_rand_discrete
[value =] nag_rand_ref_vec(seed, ref  [, optional arguments])
```

The function result is a scalar of type integer.

or

```
[value =] nag_rand_ref_vec(seed, ref, r  [, optional arguments])
```

The function returns an array-valued result of type integer and of dimension (r).

3 Arguments

Note. All array arguments are assumed-shape arrays. The extent in each dimension must be exactly that required by the problem. Notation such as '$\mathbf{x}(n)$' is used in the argument descriptions to specify that the array x must have exactly n elements.

This procedure derives the value of the following problem parameter from the shape of the supplied arrays.

r — the number of random integers to be generated

3.1 Mandatory Arguments

seed — type(nag_seed_*wp*), intent(inout)

Input: the seed for generating random numbers (see the Module Introduction).

Output: an updated value of the seed.

ref — type(nag_ref_vec_*wp*), intent(in)

Input: the reference vector.

Constraints: ref must be set up by a call to one of the procedures for generating discrete distributions.

r — integer, intent(in)

Input: the number of random numbers to be generated, if a vector-valued result is required.

Note: this argument must be omitted if a scalar result is required. For r ≤ 0 an empty array will be returned.

3.2 Optional Argument

error — type(nag_error), intent(inout), optional

The NAG *fl*90 error-handling argument. See the Essential Introduction, or the module document nag_error_handling (1.2). You are recommended to omit this argument if you are unsure how to use it. If this argument is supplied, it *must* be initialized by a call to nag_set_error before this procedure is called.

4 Error Codes

Fatal errors (error%level = 3):

error%code	Description
301	An input argument has an invalid value.

5 Examples of Usage

A complete example of the use of this procedure appears in Example 1 of this module document.

6 Further Comments

6.1 Algorithmic Detail

This procedure first generates a uniform random variate x, and then searches the CDF in the reference vector `ref` for the smallest value y such that CDF $(y) \geq x$ and CDF $(y-1) < x$.

Derived Type: nag_ref_vec_*wp*

Note. The names of derived types containing real/complex components are precision dependent. For double precision the name of this type is `nag_ref_vec_dp`. For single precision the name is `nag_ref_vec_sp`. Please read the Users' Note for your implementation to check which precisions are available.

1 Description

The derived type `nag_ref_vec_wp` stores a *reference vector* which contains information about the PDF (probability density function) or CDF (cumulative density function) of a discrete distribution, and which may be passed to the procedure `nag_rand_ref_vec` to generate random numbers from that distribution.

The components of this type are private.

The procedures which output structures of this type allocate storage to the pointer components of the structure. The amount of storage depends on the particular distribution and its parameters.

If you wish to deallocate the storage when the structure is no longer required, you must call the generic deallocation procedure which is described in the module document `nag_lib_support` (1.1).

2 Type Definition

```
type nag_ref_vec_wp
  private
  .
  .
  .
end type nag_ref_vec_wp
```

3 Components

In order to reduce the risk of accidental data corruption the components of this type are private and may not be accessed directly.

Example 1: Generation of random integers from a discrete distribution specified only in terms of its CDF

This example shows two methods of using `nag_rand_user_dist` to return random integers from a user-specified distribution where only the CDF is given. The first method involves one step while the second method uses two steps to generate random integers, but both methods return the same results.

1 Program Text

Note. The listing of the example program presented below is double precision. Single precision users are referred to Section 5.2 of the Essential Introduction for further information.

```
PROGRAM nag_rand_discrete_ex01

  ! Example Program Text for nag_rand_discrete
  ! NAG fl90, Release 3. NAG Copyright 1997.

  ! .. Use Statements ..
  USE nag_examples_io, ONLY : nag_std_in, nag_std_out
  USE nag_rand_discrete, ONLY : nag_rand_seed_set, nag_rand_user_dist, &
   nag_rand_ref_vec, nag_deallocate, nag_seed_wp => nag_seed_dp, &
   nag_ref_vec_wp => nag_ref_vec_dp
  ! .. Implicit None Statement ..
  IMPLICIT NONE
  ! .. Intrinsic Functions ..
  INTRINSIC KIND
  ! .. Parameters ..
  INTEGER, PARAMETER :: wp = KIND(1.0D0)
  CHARACTER (*), PARAMETER :: fmt = '(/1x,a/,(1X, 8i5))'
  ! .. Local Scalars ..
  INTEGER :: k, n, r, val
  LOGICAL :: pdf
  TYPE (nag_ref_vec_wp) :: ref
  TYPE (nag_seed_wp) :: seed
  ! .. Local Arrays ..
  INTEGER, ALLOCATABLE :: iv(:)
  REAL (wp), ALLOCATABLE :: p(:)
  ! .. Executable Statements ..

  WRITE (nag_std_out,*) &
   'Example Program Results for nag_rand_discrete_ex01'

  READ (nag_std_in,*)          ! skip heading in data file
  READ (nag_std_in,*) n        ! distribution size
  READ (nag_std_in,*) r        ! number of random numbers

  ALLOCATE (p(n),iv(r))        ! Allocate storage

  READ (nag_std_in,*) p        ! cdf

  ! Generating random integers in one step
  k = 0

  CALL nag_rand_seed_set(seed,k)

  val = 0
  pdf = .FALSE.

  CALL nag_rand_user_dist(p,val,pdf,iv=iv,seed=seed)

  WRITE (nag_std_out,fmt) 'User supplied CDF; one step approach', iv
```

```
      ! Generating random integers in two steps

      CALL nag_rand_user_dist(p,val,pdf,ref=ref) ! Create reference vector

      k = 0

      CALL nag_rand_seed_set(seed,k)

      iv = nag_rand_ref_vec(seed,ref,r)

      WRITE (nag_std_out,fmt) 'User supplied CDF; two steps approach', iv

      DEALLOCATE (iv,p)            ! Deallocate storage

      CALL nag_deallocate(ref)     ! Free structure allocated by NAG fl90

    END PROGRAM nag_rand_discrete_ex01
```

2 Program Data

```
Example Program Data for nag_rand_discrete_ex01
  10                                      : Number of values in CDF
  5                                       : Number of random numbers
  0.0 0.1 0.2 0.4 0.5 0.6 0.8 0.9 1.0 1.0  : CDF
```

3 Program Results

```
 Example Program Results for nag_rand_discrete_ex01

 User supplied CDF; one step approach
     6    3    3    3    7

 User supplied CDF; two steps approach
     6    3    3    3    7
```

Additional Examples

Not all example programs supplied with NAG *fl*90 appear in full in this module document. The following additional examples, associated with this module, are available.

`nag_rand_discrete_ex02`

Generation of integer random numbers from a hypergeometric distribution with known parameters and/or setting up a reference vector.

`nag_rand_discrete_ex03`

Generation of integer random numbers from a binomial distribution with known parameters and/or setting up a reference vector.

`nag_rand_discrete_ex04`

Generation of integer random numbers from a negative binomial distribution with known parameters and/or setting up a reference vector.

References

[1] Dagpunar J (1988) *Principles of Random Variate Generation* Oxford University Press

[2] Kendall M G and Stuart A (1976) *The Advanced Theory of Statistics (Volume 3)* Griffin (3rd Edition)

[3] Knuth D E (1981) *The Art of Computer Programming (Volume 2)* Addison-Wesley (2nd Edition)

[4] Morgan B J T (1984) *Elements of Simulation* Chapman and Hall

[5] Ripley B D (1987) *Stochastic Simulation* Wiley