1 Purpose

nag_rand_gen_multinomial (g05tgc) generates a sequence of \( n \) variates, each consisting of \( k \) pseudorandom integers, from the discrete multinomial distribution with \( k \) outcomes and \( m \) trials, where the outcomes have probabilities \( p_1, p_2, \ldots, p_k \) respectively.

2 Specification

```c
#include <nag.h>
#include <nagg05.h>

void nag_rand_gen_multinomial (Nag_OrderType order, Nag_ModeRNG mode,
                              Integer n, Integer m, Integer k, const double p[],
                              double r[], Integer lr, Integer state[], Integer x[],
                              Integer pdx, NagError *fail)
```

3 Description

nag_rand_gen_multinomial (g05tgc) generates a sequence of \( n \) groups of \( k \) integers \( x_{ij} \), for \( j = 1, 2, \ldots, k \) and \( i = 1, 2, \ldots, n \), from a multinomial distribution with \( m \) trials and \( k \) outcomes, where the probability of \( x_{ij} = I_j \) for each \( j = 1, 2, \ldots, k \) is

\[
P(i_1 = I_1, \ldots, i_k = I_k) = \frac{m!}{I_1! I_2! \cdots I_k!} p_1^{I_1} p_2^{I_2} \cdots p_k^{I_k},
\]

where

\[
\sum_{j=1}^{k} p_j = 1 \quad \text{and} \quad \sum_{j=1}^{k} I_j = m.
\]

A single trial can have several outcomes (\( k \)) and the probability of achieving each outcome is known (\( p_j \)). After \( m \) trials each outcome will have occurred a certain number of times. The \( k \) numbers representing the numbers of occurrences for each outcome after \( m \) trials is then a single sample from the multinomial distribution defined by the parameters \( k \), \( m \) and \( p_j \), for \( j = 1, 2, \ldots, k \). This function returns \( n \) such samples.

When \( k = 2 \) this distribution is equivalent to the binomial distribution with parameters \( m \) and \( p = p_1 \) (see nag_rand_binomial (g05tac)).

The variates can be generated with or without using a search table and index. If a search table is used then it is stored with the index in a reference vector and subsequent calls to nag_rand_gen_multinomial (g05tgc) with the same parameter values can then use this reference vector to generate further variates. The reference array is generated only for the outcome with greatest probability. The number of successes for the outcome with greatest probability is calculated first as for the binomial distribution (see nag_rand_binomial (g05tac)); the number of successes for other outcomes are calculated in turn for the remaining reduced multinomial distribution; the number of successes for the final outcome is simply calculated to ensure that the total number of successes is \( m \).

One of the initialization functions nag_rand_init_repeatable (g05kfc) (for a repeatable sequence if computed sequentially) or nag_rand_init_nonrepeatable (g05kge) (for a non-repeatable sequence) must be called prior to the first call to nag_rand_gen_multinomial (g05tgc).
4 References

5 Arguments
1: order – Nag_OrderType  
   Input
   On entry: the order argument specifies the two-dimensional storage scheme being used, i.e., row-
major ordering or column-major ordering. C language defined storage is specified by
   order = Nag_RowMajor. See Section 3.2.1.3 in the Essential Introduction for a more detailed
   explanation of the use of this argument.
   Constraint: order = Nag_RowMajor or Nag_ColMajor.

2: mode – Nag_ModeRNG  
   Input
   On entry: a code for selecting the operation to be performed by the function.
   mode = Nag.InitializeReference
      Set up reference vector only.
   mode = Nag.GenerateFromReference
      Generate variates using reference vector set up in a prior call to nag_rand_gen_multinomial
      (g05tgc).
   mode = Nag.InitializeAndGenerate
      Set up reference vector and generate variates.
   mode = Nag.GenerateWithoutReference
      Generate variates without using the reference vector.
   Constraint: mode = Nag.InitializeReference, Nag.GenerateFromReference,
   Nag.InitializeAndGenerate or Nag.GenerateWithoutReference.

3: n – Integer  
   Input
   On entry: n, the number of pseudorandom numbers to be generated.
   Constraint: n ≥ 0.

4: m – Integer  
   Input
   On entry: m, the number of trials of the multinomial distribution.
   Constraint: m ≥ 0.

5: k – Integer  
   Input
   On entry: k, the number of possible outcomes of the multinomial distribution.
   Constraint: k ≥ 2.

6: p[k] – const double  
   Input
   On entry: contains the probabilities p_j, for j = 1, 2, ..., k, of the k possible outcomes of the
   multinomial distribution.
   Constraint: 0.0 ≤ p[j - 1] ≤ 1.0 and \sum_{j=1}^{k} p[j - 1] = 1.0.

7: r[lr] – double  
   Communication Array
   On entry: if mode = Nag.GenerateFromReference, the reference vector from the previous call to
   nag_rand_gen_multinomial (g05tgc).
If mode = Nag_GenerateWithoutReference, r is not referenced and may be NULL.

On exit: if mode ≠ Nag_GenerateWithoutReference, the reference vector.

8: lr – Integer
   Input
Note: for convenience \( p_{m,ax} \) will be used here to denote the expression \( p_{m,ax} = \max_j(p[j]) \).

On entry: the dimension of the array r.

Suggested value:

if mode ≠ Nag_GenerateWithoutReference, \( lr = 30 + 20 \times \sqrt{m \times p_{m,ax} \times (1 - p_{m,ax})} \);
otherwise \( lr = 1 \).

Constraints:

if mode = Nag_InitializeReference or Nag_InitializeAndGenerate,
\[
lr > \min(m, \text{INT}\left[m \times p_{m,ax} + 7.25 \times \sqrt{m \times p_{m,ax} \times (1 - p_{m,ax}) + 8.5}\right] - 7.25 \times \sqrt{m \times p_{m,ax} \times (1 - p_{m,ax})} + 9),
\]
if mode = Nag_GenerateFromReference, lr must remain unchanged from the previous call
to nag_rand_gen_multinomial (g05tgc).

9: state[dim] – Integer
   Communication Array
Note: the dimension, dim, of this array is dictated by the requirements of associated functions that
must have been previously called. This array MUST be the same array passed as argument state
in the previous call to nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc).

On entry: contains information on the selected base generator and its current state.
On exit: contains updated information on the state of the generator.

10: x[dim] – Integer
    Output
Note: the dimension, dim, of the array x must be at least
\[
\max(1, pdx \times k) \text{ when order = Nag_ColMajor} ; \text{ max}(1, n \times pdx) \text{ when order = Nag_RowMajor}.
\]

Where \( X(i,j) \) appears in this document, it refers to the array element
\[
x[(j - 1) \times pdx + i - 1] \text{ when order = Nag_ColMajor} ; \text{ x[(i - 1) \times pdx + j - 1] when order = Nag_RowMajor}.
\]

On exit: the first \( n \) rows of \( X(i,j) \) each contain \( k \) pseudorandom numbers representing a
\( k \)-dimensional variate from the specified multinomial distribution.

11: pdx – Integer
    Input
On entry: the stride separating row or column elements (depending on the value of order) in the
array x.

Constraints:

if order = Nag_ColMajor, \( pdx \geq n \);
if order = Nag_RowMajor, \( pdx \geq k \).

12: fail – NagError *
    Input/Output
The NAG error argument (see Section 3.6 in the Essential Introduction).
6 Error Indicators and Warnings

NE_ALLOC_FAIL
Dynamic memory allocation failed.
See Section 3.2.1.2 in the Essential Introduction for further information.

NE_BAD_PARAM
On entry, argument <value> had an illegal value.

NE_INT
On entry, \( k = \langle value \rangle \).
Constraint: \( k \geq 2 \).

On entry, \( lr \) is too small when \( \text{mode} = \text{Nag\_InitializeReference} \) or \( \text{Nag\_InitializeAndGenerate} \):
\( lr = \langle value \rangle \), minimum length required = \( \langle value \rangle \).

On entry, \( m = \langle value \rangle \).
Constraint: \( m \geq 0 \).

On entry, \( n = \langle value \rangle \).
Constraint: \( n \geq 0 \).

NE_INT_2
On entry, \( pdx = \langle value \rangle \) and \( k = \langle value \rangle \).
Constraint: \( pdx \geq k \).

On entry, \( pdx = \langle value \rangle \) and \( n = \langle value \rangle \).
Constraint: \( pdx \geq n \).

NE_INTERNAL_ERROR
An internal error has occurred in this function. Check the function call and any array sizes. If the
call is correct then please contact NAG for assistance.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 3.6.6 in the Essential Introduction for further information.

NE_INVALID_STATE
On entry, \( \text{state} \) vector has been corrupted or not initialized.

NE_NO_LICENCE
Your licence key may have expired or may not have been installed correctly.
See Section 3.6.5 in the Essential Introduction for further information.

NE_PREV_CALL
The value of \( m \) or \( k \) is not the same as when \( r \) was set up in a previous call.
Previous value of \( m = \langle value \rangle \) and \( m = \langle value \rangle \).
Previous value of \( k = \langle value \rangle \) and \( k = \langle value \rangle \).

NE_REAL_ARRAY
On entry, at least one element of the vector \( p \) is less than 0 or greater than 1.
On entry, the sum of the elements of \( p \) do not equal one.

NE_REF_VEC
On entry, some of the elements of the array \( r \) have been corrupted or have not been initialized.
7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_rand_gen_multinomial (g05tgc) is threaded by NAG for parallel execution in multithreaded implementations of the NAG Library.

Please consult the X06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users’ Note for your implementation for any additional implementation-specific information.

9 Further Comments

The reference vector for only one outcome can be set up because the conditional distributions cannot be known in advance of the generation of variates. The outcome with greatest probability of success is chosen for the reference vector because it will have the greatest spread of likely values.

10 Example

This example prints 20 pseudorandom $k$-dimensional variates from a multinomial distribution with $k = 4$, $m = 6000$, $p_1 = 0.08$, $p_2 = 0.1$, $p_3 = 0.8$ and $p_4 = 0.02$, generated by a single call to nag_rand_gen_multinomial (g05tgc), after initialization by nag_rand_init_repeatable (g05kfc).

10.1 Program Text

/* nag_rand_gen_multinomial (g05tgc) Example Program.  *
 * Copyright 2014 Numerical Algorithms Group.  *
 * Mark 9, 2009.  */

/* Pre-processor includes */
#include <stdio.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

#define X(I, J) x[(order == Nag_ColMajor)?(J*pdx + I):(I*pdx + J)]

int main(void)
{
    /* Integer scalar and array declarations */
    Integer exit_status = 0;
    Integer lr, x_size, i, j, lstate, pdx;
    Integer *state = 0, *x = 0;
    /* NAG structures */
    NagError fail;
    Nag_ModeRNG mode;

    /* Double scalar and array declarations */
    double p_max;
    double *r = 0;

    /* Set the distribution parameters */
    Integer k = 4;
    Integer m = 6000;
    double p[] = { 0.08e0, 0.1e0, 0.8e0, 0.02e0 };

    /* Set the sample size */
    Integer n = 20;
/* Return the results in column major order */
Nag_OrderType order = Nag_ColMajor;

/* Choose the base generator */
Nag_BaseRNG genid = Nag_Basic;
Integer subid = 0;

/* Set the seed */
Integer seed[] = { 1762543 };
Integer lseed = 1;

/* Initialise the error structure */
INIT_FAIL(fail);

printf(
    "nag_rand_gen_multinomial (g05tgc) Example Program Results\n\n"
);

/* Get the length of the state array */
lstate = -1;
nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}
pdx = (order == Nag_ColMajor)?n:k;
x_size = (order == Nag_ColMajor)?pdx * k:pdx * n;

/* Calculate the size of the reference vector */
p_max = 0.0;
for (i = 1; i < k; i++)
    p_max = (p_max < p[i])?p[i]:p_max;
lr = 30 + 20 * sqrt(m * p_max * (1 - p_max));

/* Allocate arrays */
if (!(r = NAG_ALLOC(lr, double)) ||
    !(state = NAG_ALLOC(lstate, Integer)) ||
    !(x = NAG_ALLOC(x_size, Integer)))
{
    printf("Allocation failure\n");
    exit_status = -1;
    goto END;
}

/* Initialise the generator to a repeatable sequence */
nag_rand_init_repeatable(genid, subid, seed, lseed, state, &lstate, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_init_repeatable (g05kfc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}

/* Generate the variates, initialising the reference vector */
mode = Nag_InitializeAndGenerate;
nag_rand_gen_multinomial(order, mode, n, m, k, p, r, lr, state, x, pdx,
    &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_rand_gen_multinomial (g05tgc).\n%s\n",
        fail.message);
    exit_status = 1;
    goto END;
}
/* Display the variates*/
for (i = 0; i < n; i++)
{
    for (j = 0; j < k; j++)
        printf("%12"\"NAG_IFMT\"", X(i, j));
    printf("\n");
}

END:
NAG_FREE(r);
NAG_FREE(state);
NAG_FREE(x);

return exit_status;

10.2 Program Data
None.

10.3 Program Results
nag_rand_gen_multinomial (g05tgc) Example Program Results

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>468</td>
<td>603</td>
<td>4811</td>
<td>118</td>
</tr>
<tr>
<td>490</td>
<td>630</td>
<td>4761</td>
<td>119</td>
</tr>
<tr>
<td>482</td>
<td>575</td>
<td>4821</td>
<td>122</td>
</tr>
<tr>
<td>495</td>
<td>591</td>
<td>4826</td>
<td>88</td>
</tr>
<tr>
<td>512</td>
<td>611</td>
<td>4761</td>
<td>116</td>
</tr>
<tr>
<td>474</td>
<td>601</td>
<td>4800</td>
<td>125</td>
</tr>
<tr>
<td>485</td>
<td>595</td>
<td>4791</td>
<td>129</td>
</tr>
<tr>
<td>468</td>
<td>582</td>
<td>4825</td>
<td>125</td>
</tr>
<tr>
<td>485</td>
<td>598</td>
<td>4800</td>
<td>117</td>
</tr>
<tr>
<td>485</td>
<td>573</td>
<td>4814</td>
<td>128</td>
</tr>
<tr>
<td>501</td>
<td>634</td>
<td>4749</td>
<td>116</td>
</tr>
<tr>
<td>482</td>
<td>618</td>
<td>4780</td>
<td>120</td>
</tr>
<tr>
<td>470</td>
<td>584</td>
<td>4810</td>
<td>136</td>
</tr>
<tr>
<td>479</td>
<td>642</td>
<td>4750</td>
<td>129</td>
</tr>
<tr>
<td>476</td>
<td>608</td>
<td>4807</td>
<td>109</td>
</tr>
<tr>
<td>473</td>
<td>631</td>
<td>4782</td>
<td>114</td>
</tr>
<tr>
<td>509</td>
<td>596</td>
<td>4778</td>
<td>117</td>
</tr>
<tr>
<td>450</td>
<td>565</td>
<td>4877</td>
<td>108</td>
</tr>
<tr>
<td>484</td>
<td>556</td>
<td>4840</td>
<td>120</td>
</tr>
<tr>
<td>466</td>
<td>615</td>
<td>4802</td>
<td>117</td>
</tr>
</tbody>
</table>