

# NAG Library Function Document

## nag\_rand\_sample\_unequal (g05nec)

### 1 Purpose

nag\_rand\_sample\_unequal (g05nec) selects a pseudorandom sample, without replacement and allowing for unequal probabilities.

### 2 Specification

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

void nag_rand_sample_unequal (Nag_SortOrder sortorder, const double wt[],
    const Integer ipop[], Integer n, Integer isamp1[], Integer m,
    Integer state[], NagError *fail)
```

### 3 Description

nag\_rand\_sample\_unequal (g05nec) selects  $m$  elements from either the set of values  $(1, 2, \dots, n)$  or a supplied population vector of length  $n$ . The probability of selecting the  $i$ th element is proportional to a user-supplied weight,  $w_i$ . Each element will appear at most once in the sample, i.e., the sampling is done without replacement.

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\_sample\_unequal (g05nec).

### 4 References

None.

### 5 Arguments

1: **sortorder** – Nag\_SortOrder *Input*

*On entry:* a flag indicating the sorted status of the **wt** vector.

**sortorder** = Nag\_Ascending  
**wt** is sorted in ascending order,

**sortorder** = Nag\_Descending  
**wt** is sorted in descending order,

**sortorder** = Nag\_Unsorted  
**wt** is unsorted and nag\_rand\_sample\_unequal (g05nec) will sort the weights prior to using them.

Irrespective of the value of **sortorder**, no checks are made on the sorted status of **wt**, e.g., it is possible to supply **sortorder** = Nag\_Ascending, even when **wt** is not sorted. In such cases the **wt** array will not be sorted internally, but nag\_rand\_sample\_unequal (g05nec) will still work correctly except, possibly, in cases of extreme weight values.

It is usually more efficient to specify a value of **sortorder** that is consistent with the status of **wt**.

*Constraint:* **sortorder** = Nag\_Ascending, Nag\_Descending or Nag\_Unsorted.

- 2: **wt[n]** – const double *Input*  
*On entry:*  $w_i$ , the relative probability weights. These weights need not sum to 1.0.  
*Constraints:*  
 $\text{wt}[i - 1] \geq 0.0$ , for  $i = 1, 2, \dots, \mathbf{n}$ ;  
at least **m** values must be nonzero.
- 3: **ipop[dim]** – const Integer *Input*  
**Note:** the dimension, *dim*, of the array **ipop** must be at least **n** when **ipop** is not **NULL**.  
*On entry:* the population to be sampled. If **ipop** is **NULL** then the population is assumed to be the set of values  $(1, 2, \dots, \mathbf{n})$  and the array **ipop** is not referenced. Elements of **ipop** with the same value are not combined, therefore if  $\text{wt}[i - 1] \neq 0, \text{wt}[j - 1] \neq 0$  and  $i \neq j$  then there is a nonzero probability that the sample will contain both **ipop**[*i* - 1] and **ipop**[*j* - 1]. If **ipop**[*i* - 1] = **ipop**[*j* - 1] then that value can appear in **isampl** more than once.
- 4: **n** – Integer *Input*  
*On entry:* *n*, the size of the population.  
*Constraint:*  $\mathbf{n} \geq 1$ .
- 5: **isampl[m]** – Integer *Output*  
*On exit:* the selected sample.
- 6: **m** – Integer *Input*  
*On entry:* *m*, the size of the sample required.  
*Constraint:*  $0 \leq \mathbf{m} \leq \mathbf{n}$ .
- 7: **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` (g05kge).  
*On entry:* contains information on the selected base generator and its current state.  
*On exit:* contains updated information on the state of the generator.
- 8: **fail** – NagError \* *Input/Output*  
The NAG error argument (see Section 2.7 in How to Use the NAG Library and its Documentation).

## 6 Error Indicators and Warnings

### NE\_ALLOC\_FAIL

Dynamic memory allocation failed.

See Section 3.2.1.2 in How to Use the NAG Library and its Documentation for further information.

### NE\_BAD\_PARAM

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

**NE\_INT**

On entry,  $\mathbf{n} = \langle value \rangle$ .  
 Constraint:  $\mathbf{n} \geq 1$ .

**NE\_INT\_2**

On entry,  $\mathbf{m} = \langle value \rangle$  and  $\mathbf{n} = \langle value \rangle$ .  
 Constraint:  $0 \leq \mathbf{m} \leq \mathbf{n}$ .

**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 How to Use the NAG Library and its Documentation for further information.

**NE\_INVALID\_STATE**

On entry, **state** vector has been corrupted or not initialized.

**NE\_NEG\_WEIGHT**

On entry, at least one weight was less than zero.

**NE\_NO\_LICENCE**

Your licence key may have expired or may not have been installed correctly.  
 See Section 3.6.5 in How to Use the NAG Library and its Documentation for further information.

**NE\_NON\_ZERO\_WEIGHTS**

On entry,  $\mathbf{m} = \langle value \rangle$ , number of nonzero weights =  $\langle value \rangle$ .  
 Constraint: must be at least  $\mathbf{m}$  nonzero weights.

**7 Accuracy**

Not applicable.

**8 Parallelism and Performance**

nag\_rand\_sample\_unequal (g05nec) 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**

nag\_rand\_sample\_unequal (g05nec) internally allocates  $(\mathbf{n} + 1)$  doubles and  $\mathbf{n}$  Integers.

Although it is possible to use nag\_rand\_sample\_unequal (g05nec) to sample using equal probabilities, by setting all elements of the input array **wt** to the same positive value, it is more efficient to use nag\_rand\_sample (g05ndc). To sample with replacement, nag\_rand\_gen\_discrete (g05tdc) can be used when the probabilities are unequal and nag\_rand\_discrete\_uniform (g05tlc) when the probabilities are equal.

**10 Example**

This example samples from a population of 25.

## 10.1 Program Text

```

/* nag_rand_sample_unequal (g05nec) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <nag.h>
#include <nag_stdlib.h>
#include <nagg05.h>

int main(void)
{
    /* Scalars */
    Integer exit_status = 0, lseed = 1;
    Integer i, lstate, m, n, subid;

    /* Arrays */
    Integer *ipop = 0, *isAMPL = 0, *state = 0;
    Integer seed[1];
    double *wt = 0;
    char cgenid[40], csortorder[40], cpop_supplied[40];

    /* NAG structures */
    NagError fail;
    Nag_BaseRNG genid;
    Nag_SortOrder sortorder;
    Nag_Boolean pop_supplied;

    /* Initialize the error structure to print out any error messages */
    INIT_FAIL(fail);

    printf("nag_rand_sample_unequal (g05nec) Example Program Results\n\n");

    /* Skip heading in data file */
#ifdef _WIN32
    scanf_s("%*[\n] ");
#else
    scanf("%*[\n] ");
#endif

    /* Read in the base generator information and seed */
#ifdef _WIN32
    scanf_s("%39s% NAG_IFMT %" NAG_IFMT "%*[\n] ", cgenid,
            (unsigned)_countof(cgenid), &subid, &seed[0]);
#else
    scanf("%39s% NAG_IFMT %" NAG_IFMT "%*[\n] ", cgenid, &subid, &seed[0]);
#endif
    genid = (Nag_BaseRNG) nag_enum_name_to_value(cgenid);

    /* Query to obtain the length of the state array using
     * nag_rand_init_repeatable (g05kfc).
     */
    lstate = 0;
    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;
    }

    /* Allocate memory to state */
    if (!(state = NAG_ALLOC(lstate, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -1;
    }
}

```

```

    goto END;
}

/* Initialize the RNG using
 * nag_rand_init_repeatable (g05kfc)
 */
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 = 2;
    goto END;
}

/* Read in the problem size, pop_supplied is a True / False flag indicating
 * whether population flags are supplied (Nag_TRUE) or taken as the integers
 * 1 to n (Nag_FALSE)
 */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%39s%39s%[^\\n] ", &n, &m, csortorder,
        (unsigned)_countof(csortorder), cpop_supplied,
        (unsigned)_countof(cpop_supplied));
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%39s%39s%[^\\n] ", &n, &m, csortorder,
        cpop_supplied);
#endif
sortorder = (Nag_SortOrder) nag_enum_name_to_value(csortorder);
pop_supplied = (Nag_Boolean) nag_enum_name_to_value(cpop_supplied);

/* Allocate memory for input arrays */
if (!(wt = NAG_ALLOC(n, double)) || !(isampl = NAG_ALLOC(m, Integer))
    )
{
    printf("Allocation failure\n");
    exit_status = -2;
    goto END;
}

if (pop_supplied) {
    /* Read in the population and weights */
    if (!(ipop = NAG_ALLOC(n, Integer)))
    {
        printf("Allocation failure\n");
        exit_status = -3;
        goto END;
    }
}
#ifdef _WIN32
    for (i = 0; i < n; i++)
        scanf_s("%" NAG_IFMT "%lf%[^\\n] ", &ipop[i], &wt[i]);
#else
    for (i = 0; i < n; i++)
        scanf("%" NAG_IFMT "%lf%[^\\n] ", &ipop[i], &wt[i]);
#endif
}
else {
    /* Read in just the weights */
#ifdef _WIN32
    for (i = 0; i < n; i++)
        scanf_s("%lf%[^\\n] ", &wt[i]);
#else
    for (i = 0; i < n; i++)
        scanf("%lf%[^\\n] ", &wt[i]);
#endif
}

/* Generate the sample without replacement, unequal weights using
 * nag_rand_sample_unequal (g05nec)
 */
nag_rand_sample_unequal(sortorder, wt, ipop, n, isampl, m, state, &fail);
if (fail.code != NE_NOERROR) {
    printf("Error from nag_rand_sample_unequal (g05nec).\n%s\n",

```

```

        fail.message);
    exit_status = 3;
    goto END;
}

/* Display the results */
for (i = 0; i < m; i++)
    printf("%5" NAG_IFMT " ", isampl[i]);
printf("\n");

END:
    NAG_FREE(wt);
    NAG_FREE(ipop);
    NAG_FREE(isampl);
    NAG_FREE(state);

    return exit_status;
}

```

## 10.2 Program Data

```

nag_rand_sample_unequal (g05nec) Example Program Data
Nag_MersenneTwister      0 1762543 :: genid, subid, seed[0]
 25  10  Nag_Unsorted Nag_TRUE :: n, m, sortorder, pop_supplied
171  85.54
 52  71.78
172 118.13
139  13.68
196 153.60
125 165.35
 36 122.35
 70  35.87
 25 151.78
 86 128.33
 76 178.27
 37 183.37
185 165.81
 40 101.41
 90 145.16
 27  42.01
 79  59.08
118  17.53
142  87.14
127  69.20
101  31.13
 22  60.26
 41  21.00
199  85.06
 59 119.73                                :: End of ipop,wt

```

## 10.3 Program Results

nag\_rand\_sample\_unequal (g05nec) Example Program Results

```

125  41 185  40  37 196  22  25  76 172

```

---