

NAG Library

Advice on Replacement Calls for Withdrawn/Superseded Functions

The following list gives the names of replacement functions for those functions that have been withdrawn or superseded.

The list indicates the minimum change necessary, but many of the replacement functions have additional flexibility and you may wish to take advantage of new features. It is strongly recommended that you consult the function documents.

a02 – Complex Arithmetic

nag_complex_sqrt (a02aac)

Withdrawn at Mark 2.

There is no replacement for this function.

nag_complex_sqrt (a02abc)

Withdrawn at Mark 2.

There is no replacement for this function.

nag_complex_divide (a02acc)

Withdrawn at Mark 2.

There is no replacement for this function.

c05 – Roots of One or More Transcendental Equations

nag_zero_cont_func_bd (c05adc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_zero_cont_func_bd_1 (c05sdc).

nag_zero_nonlin_eqns (c05nbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_zero_nonlin_eqns_1 (c05tbc).

nag_zero_nonlin_eqns_deriv (c05pbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_zero_nonlin_eqns_deriv_1 (c05ubc).

nag_check_deriv (c05zbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_check_deriv_1 (c05zcc).

d01 – Quadrature

nag_1d_quad_gen (d01ajc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_1d_quad_gen_1 (d01sjc).

nag_1d_quad_osc (d01akc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_1d_quad_osc_1 (d01skc).

nag_1d_quad_brkpts (d01alc)

Scheduled for withdrawal at Mark 11.
Replaced by nag_1d_quad_brkpts_1 (d01slc).

nag_1d_quad_inf (d01amc)

Scheduled for withdrawal at Mark 11.
Replaced by nag_1d_quad_inf_1 (d01smc).

nag_1d_quad_wt_alglog (d01apc)

Scheduled for withdrawal at Mark 11.
Replaced by nag_1d_quad_wt_alglog_1 (d01spc).

nag_1d_quad_wt_cauchy (d01aqc)

Scheduled for withdrawal at Mark 11.
Replaced by nag_1d_quad_wt_cauchy_1 (d01sqc).

nag_1d_quad_inf_wt_trig (d01asc)

Scheduled for withdrawal at Mark 11.
Replaced by nag_1d_quad_inf_wt_trig_1 (d01ssc).

nag_1d_quad_gauss (d01bac)

Scheduled for withdrawal at Mark 11.
Replaced by nag_1d_quad_gauss_1 (d01tac).

nag_multid_quad_adapt (d01fcc)

Scheduled for withdrawal at Mark 11.
Replaced by nag_multid_quad_adapt_1 (d01wcc).

nag_multid_quad_monte_carlo (d01gbc)

Scheduled for withdrawal at Mark 11.
Replaced by nag_multid_quad_monte_carlo_1 (d01xbc).

e01 – Interpolation**nag_2d_scatter_interpolant (e01sac)**

Scheduled for withdrawal at Mark 10.
Replaced by nag_2d_shep_interp (e01sgc) or nag_2d_triang_interp (e01sjc).

nag_2d_scatter_interpolant (e01sac) generates a two-dimensional surface interpolating a set of scattered data points, using either the method of Renka and Cline or a modification of Shepard's method. The replacement functions separate these two methods. e01sac_rk.c (see http://www.nag.co.uk/numerical/CL/nagdoc_cl09/xhtml/examples/replaced/e01sac_rk.c) provides replacement call information for the Renka and Cline method (nag_2d_shep_interp (e01sgc)) and e01sac_shep.c (see http://www.nag.co.uk/numerical/CL/nagdoc_cl09/xhtml/examples/replaced/e01sac_shep.c) provides replacement call information for the Shepard's method (nag_2d_triang_interp (e01sjc)).

nag_2d_scatter_eval (e01sbc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_2d_shep_eval (e01shc) or nag_2d_triang_eval (e01skc).

See the example program e01sac_rk.c (see http://www.nag.co.uk/numerical/CL/nagdoc_cl09/xhtml/examples/replaced/e01sac_rk.c) and e01sac_shep.c (see http://www.nag.co.uk/numerical/CL/nagdoc_cl09/xhtml/examples/replaced/e01sac_shep.c) for full details.

nag_2d_scatter_free (e01szc)

Scheduled for withdrawal at Mark 10.
There is no replacement for this function.

e04 – Minimizing or Maximizing a Function

nag_opt_simplex (e04ccc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_opt_simplex_easy (e04cbc).

The new function can be derived from the old as follows:

```
Old: nag_opt_simplex(n, funct, x, &objf, &options, &comm, &fail);
New: nag_opt_simplex_easy(n, x, &objf, tolf, tolx, funct, monit, maxcal,
    &comm, &fail);
```

The options structure has been removed from nag_opt_simplex (e04ccc). The **optim_tol** member of the structure has been introduced as the argument **tolf**. **tolx** is an additional argument to control tolerance.

nag_opt_check_deriv (e04hcc)

Scheduled for withdrawal at Mark 10.

There is no replacement for this function.

nag_opt_check_deriv (e04hcc) was used to check that a user-supplied objective function, as used by nag_opt_bounds_deriv (e04kbc), returned derivatives which were consistent with the objective function itself. This function is no longer required. Similar functionality can be obtained using the optional argument **verify_grad** of nag_opt_nlp (e04ucc) or **Verify Level** of nag_opt_nlp_solve (e04wdc). See the function documents for further information.

nag_opt_check_2nd_deriv (e04hdc)

Scheduled for withdrawal at Mark 10.

There is no replacement for this function.

nag_opt_bounds_no_deriv (e04jbc)

Scheduled for withdrawal at Mark 10.

Replaced by nag_opt_nlp_solve (e04wdc).

See the example program e04jbce.c (see http://www.nag.co.uk/numerical/CL/nagdoc_cl09/xhtml/examples/replaced/e04jbce.c) for full details.

nag_opt_bounds_deriv (e04kbc)

Scheduled for withdrawal at Mark 10.

Replaced by nag_opt_nlp_solve (e04wdc).

See the example program e04kbce.c (see http://www.nag.co.uk/numerical/CL/nagdoc_cl09/xhtml/examples/replaced/e04kbce.c) for full details.

f06 – Linear Algebra Support Functions

old_dgemv (f06pac)

Scheduled for withdrawal at Mark 10.

Replaced by nag_dgemv (f16pac).

old_dgbmv (f06pbc)

Scheduled for withdrawal at Mark 10.

Replaced by nag_dgbmv (f16pbc).

old_dsymv (f06pcc)

Scheduled for withdrawal at Mark 10.

Replaced by nag_dsymv (f16pcc).

old_dsmbv (f06pdc)

Scheduled for withdrawal at Mark 10.

Replaced by nag_dsmbv (f16pdc).

old_dspmv (f06pec)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dspmv (f16pec).

old_dtrmv (f06pfc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dtrmv (f16pfc).

old_dtbmv (f06pgc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dtbmv (f16pgc).

old_dtpmv (f06phc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dtpmv (f16phc).

old_dtrsv (f06pjc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dtrsv (f16pjc).

old_dtbsv (f06pkc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dtbsv (f16pkc).

old_dtpsv (f06plc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dtpsv (f16plc).

old_dger (f06pmc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dger (f16pmc).

old_dsyv (f06ppc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dsyv (f16ppc).

old_dspr (f06pqc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dspr (f16pqc).

old_dsyv2 (f06prc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dsyv2 (f16prc).

old_dspr2 (f06psc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dspr2 (f16psc).

old_zgemv (f06sac)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zgemv (f16sac).

old_zgbmv (f06sbc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zgbmv (f16sbc).

old_zhemv (f06scc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zhemv (f16scc).

old_zhbmw (f06sdc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zhbmw (f16sdc).

old_zhpmv (f06sec)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zhpmv (f16sec).

old_ztrmv (f06sfc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_ztrmv (f16sfc).

old_ztbmv (f06sgc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_ztbmv (f16sgc).

old_ztpmv (f06shc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_ztpmv (f16shc).

old_ztrsv (f06sjc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_ztrsv (f16sjc).

old_ztbsv (f06skc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_ztbsv (f16skc).

old_ztpsv (f06slc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_ztpsv (f16slc).

old_zgeru (f06smc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zger (f16smc).

old_zgerc (f06snc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zger (f16smc).

old_zher (f06spc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zher (f16spc).

old_zhpr (f06sqc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zhpr (f16sqc).

old_zher2 (f06src)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zher2 (f16src).

old_zhpr2 (f06ssc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zhpr2 (f16ssc).

old_dgemm (f06yac)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dgemm (f16yac).

old_dsymm (f06ycc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dsymm (f16ycc).

old_dtrmm (f06yfc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dtrmm (f16yfc).

old_dtrsm (f06yjc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dtrsm (f16yjc).

old_dsyrk (f06ypc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dsyrk (f16ypc).

old_dsy2k (f06yrc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_dsy2k (f16yrc).

old_zgemm (f06zac)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zgemm (f16zac).

old_zhemm (f06zcc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zhemm (f16zcc).

old_ztrmm (f06zfc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_ztrmm (f16zfc).

old_ztrsm (f06zjc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_ztrsm (f16zjc).

old_zherk (f06zpc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zherk (f16zpc).

old_zher2k (f06zrc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zher2k (f16zrc).

old_zsymm (f06ztc)

Scheduled for withdrawal at Mark 10.
Replaced by nag_zsymm (f16ztc).

old_zsyrk (f06zuc)

Scheduled for withdrawal at Mark 10.

Replaced by nag_zsyrk (f16zuc).

old_zsyr2k (f06zwc)

Scheduled for withdrawal at Mark 10.

Replaced by nag_zsyr2k (f16zwc).

g01 – Simple Calculations on Statistical Data**nag_deviates_normal_dist (g01cec)**

Scheduled for withdrawal at Mark 11.

Replaced by nag_deviates_normal (g01fac).

g05 – Random Number Generators**nag_random_continuous_uniform (g05cac)**

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_basic (g05sac).

```
Old:
/* nag_random_continuous_uniform (g05cac) */
for (i = 0; i < n; i++)
    x[i] = nag_random_continuous_uniform();
New:
/* nag_rand_basic (g05sac) */
nag_rand_basic(n, state, x, &fail);
```

The integer array **state** in the call to nag_rand_basic (g05sac) contains information on the base generator being used. This array must have been initialized prior to calling nag_rand_basic (g05sac) with a call to either nag_rand_init_repeatabe (g05kfc) or nag_rand_init_nonrepeatabe (g05kgc). The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by nag_rand_basic (g05sac) is likely to be different from those produced by nag_random_continuous_uniform (g05cac).

nag_random_init_repeatabe (g05cbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_init_repeatabe (g05kfc).

```
Old:
/* nag_random_init_repeatabe (g05cbc) */
nag_random_init_repeatabe(i);
New:
lseed = 1;
seed[0] = i;
genid = Nag_Basic;
subid = 1;

/* nag_rand_init_repeatabe (g05kfc) */
nag_rand_init_repeatabe(genid, subid, seed, lseed, state, &lstate, &fail);
```

The integer array **state** in the call to nag_rand_init_repeatabe (g05kfc) contains information on the base generator being used. The base generator is chosen via the integer arguments **genid** and **subid**. The required length of the array **state** depends on the base generator chosen. Due to changes in the underlying code a sequence of values produced by using a random number generator initialized via a call to nag_rand_init_repeatabe (g05kfc) is likely to be different to a sequence produced by a generator initialized by nag_random_init_repeatabe (g05cbc), even if the same value for **i** is used.

nag_random_init_nonrepeatable (g05ccc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_init_nonrepeatable (g05kgc).

```

Old:
    /* nag_random_init_nonrepeatable (g05ccc) */
    nag_random_init_nonrepeatable();
New:
    genid = Nag_Basic;
    subid = 1;

    /* nag_rand_init_nonrepeatable (g05kgc) */
    nag_rand_init_nonrepeatable(genid,subid,state,&lstate,&fail);

```

The integer array **state** in the call to nag_rand_init_nonrepeatable (g05kgc) contains information on the base generator being used. The base generator is chosen via the integer arguments **genid** and **subid**. The required length of the array **state** depends on the base generator chosen.

nag_save_random_state (g05cfc)

Scheduled for withdrawal at Mark 11.

There is no replacement for this function.

```

Old:
    /* nag_save_random_state (g05cfc) */
    nag_save_random_state(istate,xstate);
New:
    for (i = 0; i < lstate; i++)
        istate[i] = state[i];

```

The state of the base generator for the group of functions nag_rand_init_repeatable (g05kfc), nag_rand_init_nonrepeatable (g05kgc), nag_rand_leap_frog (g05khc), nag_rand_skip_ahead (g05kjc), nag_rand_permute (g05ncc), nag_rand_sample (g05ndc), nag_rand_agarchI (g05pdc)–nag_rand_2_way_table (g05pzc), nag_rand_copula_students_t (g05rcc)–nag_rand_matrix_multi_normal (g05rzc), g05s and g05t can be saved by simply creating a local copy of the array **state**. The first element of the **state** array contains the number of elements that are used by the random number generating functions, therefore either this number of elements can be copied, or the whole array (as defined in the calling program).

nag_restore_random_state (g05cgc)

Scheduled for withdrawal at Mark 11.

There is no replacement for this function.

```

Old:
    /* nag_restore_random_state (g05cgc) */
    nag_restore_random_state(istate,xstate,&fail);
New:
    for (i = 0; i < lstate; i++)
        state[i] = istate[i];

```

The state of the base generator for the group of functions nag_rand_init_repeatable (g05kfc), nag_rand_init_nonrepeatable (g05kgc), nag_rand_leap_frog (g05khc), nag_rand_skip_ahead (g05kjc), nag_rand_permute (g05ncc), nag_rand_sample (g05ndc), nag_rand_agarchI (g05pdc)–nag_rand_2_way_table (g05pzc), nag_rand_copula_students_t (g05rcc)–nag_rand_matrix_multi_normal (g05rzc), g05s and g05t can be restored by simply copying back the previously saved copy of the **state** array. The first element of the **state** array contains the number of elements that are used by the random number generating functions, therefore either this number of elements can be copied, or the whole array (as defined in the calling program).

nag_random_continuous_uniform_ab (g05dac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_uniform (g05sqc).

```

Old:
    for (i = 0; i < n; i++)
        /* nag_random_continuous_uniform_ab (g05dac) */
        x[i] = nag_random_continuous_uniform_ab(aa,bb);

```

```

New:
a = (aa < bb) ? aa : bb;
b = (aa < bb) ? bb : aa;

/* nag_rand_uniform (g05sqc) */
nag_rand_uniform(n,a,b,state,x,&fail);

```

The old function `nag_random_continuous_uniform_ab (g05dac)` returns a single variate at a time, whereas the new function `nag_rand_uniform (g05sqc)` returns a vector of **n** values in one go. In `nag_rand_uniform (g05sqc)` the minimum value must be held in the argument **a** and the maximum in argument **b**, therefore **a < b**. This was not the case for the equivalent arguments in `nag_random_continuous_uniform_ab (g05dac)`.

The integer array **state** in the call to `nag_rand_uniform (g05sqc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_uniform (g05sqc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_uniform (g05sqc)` is likely to be different from those produced by `nag_random_continuous_uniform_ab (g05dac)`.

nag_random_exp (g05dbc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_exp (g05sfc)`.

```

Old:
for (i = 0; i < n; i++)
  /* nag_random_exp (g05dbc) */
  x[i] = nag_random_exp(aa);

```

```

New:
a = fabs(aa);

/* nag_rand_exp (g05sfc) */
nag_rand_exp(n,a,state,x,&fail);

```

The old function `nag_random_exp (g05dbc)` returns a single variate at a time, whereas the new function `nag_rand_exp (g05sfc)` returns a vector of **n** values in one go. In `nag_rand_exp (g05sfc)` argument **a** must be non-negative, this was not the case for the equivalent argument in `nag_random_exp (g05dbc)`.

The integer array **state** in the call to `nag_rand_exp (g05sfc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_exp (g05sfc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_exp (g05sfc)` is likely to be different from those produced by `nag_random_exp (g05dbc)`.

nag_random_normal (g05ddc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_normal (g05skc)`.

```

Old:
for (i = 0; i < n; i++)
  /* nag_random_normal (g05ddc) */
  x[i] = nag_random_normal(xmu,sd);

```

```

New:
/* nag_rand_normal (g05skc) */
nag_rand_normal(n,xmu,var,state,x,&fail);

```

The old function `nag_random_normal (g05ddc)` returns a single variate at a time, whereas the new function `nag_rand_normal (g05skc)` returns a vector of **n** values in one go. `nag_rand_normal (g05skc)` expects the variance of the Normal distribution (argument **var**), compared to `nag_random_normal (g05ddc)` which expected the standard deviation.

The integer array **state** in the call to `nag_rand_normal (g05skc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_normal (g05skc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the

underlying code the sequence of values produced by `nag_rand_normal (g05skc)` is likely to be different from those produced by `nag_random_normal (g05ddc)`.

nag_random_discrete_uniform (g05dyc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_discrete_uniform (g05tlc)`.

```
Old:
    for (i = 0; i < n; i++)
        /* nag_random_discrete_uniform (g05dyc) */
        x[i] = nag_random_discrete_uniform(aa,bb);
New:
    a = (aa < bb) ? aa : bb;
    b = (aa < bb) ? bb : aa;
    /* nag_rand_discrete_uniform (g05tlc) */
    nag_rand_discrete_uniform(n,a,b,state,x,&fail);
```

The old function `nag_random_discrete_uniform (g05dyc)` returns a single variate at a time, whereas the new function `nag_rand_discrete_uniform (g05tlc)` returns a vector of **n** values in one go. In `nag_rand_discrete_uniform (g05tlc)` the minimum value must be held in the argument **a** and the maximum in argument **b**, therefore $a \leq b$. This was not the case for the equivalent arguments in `nag_random_discrete_uniform (g05dyc)`.

The integer array **state** in the call to `nag_rand_discrete_uniform (g05tlc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_discrete_uniform (g05tlc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_discrete_uniform (g05tlc)` is likely to be different from those produced by `nag_random_discrete_uniform (g05dyc)`.

nag_ref_vec_multi_normal (g05eac)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_matrix_multi_normal (g05rzc)`.

```
Old:
    /* nag_ref_vec_multi_normal (g05eac) */
    nag_ref_vec_multi_normal(a,m,c,tdc,eps,&r,&fail);
New:
    order = Nag_RowMajor;
    mode = Nag_InitializeReference;
    lr = m * (m + 1) + 1;
    r = NAG_ALLOC(lr,double);

    /* nag_rand_matrix_multi_normal (g05rzc) */
    nag_rand_matrix_multi_normal(order,mode,n,m,a,c,tdc,r,lr,
        state,x,pdx,&fail);
```

The old function `nag_ref_vec_multi_normal (g05eac)` sets up a reference vector for use by `nag_return_multi_normal (g05ezc)`. The functionality of both these functions has been combined into the single new function `nag_rand_matrix_multi_normal (g05rzc)`. Setting **mode** = `Nag_InitializeReference` in the call to `nag_rand_matrix_multi_normal (g05rzc)` only sets up the double reference vector **r** and hence mimics the functionality of `nag_ref_vec_multi_normal (g05eac)`.

The length of the double reference vector, **r**, in `nag_rand_matrix_multi_normal (g05rzc)` must be at least $m \times (m + 1) + 1$. This is longer than the equivalent argument in `nag_ref_vec_multi_normal (g05eac)`.

nag_ref_vec_poisson (g05ecc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_poisson (g05tjc)`.

```
Old:
    /* nag_ref_vec_poisson (g05ecc) */
    nag_ref_vec_poisson(t,&r,&fail);
    for (i = 0; i < n; i++)
        /* nag_return_discrete (g05eyc) */
        x[i] = nag_return_discrete(r);
```

```

New:
mode = Nag_InitializeAndGenerate;
lr = 30 + (Integer) (20 * sqrt(t) + t);
r = NAG_ALLOC(lr, double);

/* nag_rand_poisson (g05tjc) */
nag_rand_poisson(mode, n, t, r, lr, state, x, &fail);

```

The old function `nag_ref_vec_poisson (g05ecc)` sets up a reference vector for use by `nag_return_discrete (g05eyc)`. The replacement function `nag_rand_poisson (g05tjc)` is now used to both set up a reference vector and generate the required variates. Setting **mode** = `Nag_InitializeReference` in the call to `nag_rand_poisson (g05tjc)` sets up the double reference vector **r** and hence mimics the functionality of `nag_ref_vec_poisson (g05ecc)`. Setting **mode** = `Nag_GenerateFromReference` generates a series of variates from a reference vector mimicking the functionality of `nag_return_discrete (g05eyc)` for this particular distribution. Setting **mode** = `Nag_InitializeAndGenerate` initializes the reference vector and generates the variates in one go.

The function `nag_return_discrete (g05eyc)` returns a single variate at a time, whereas the new function `nag_rand_poisson (g05tjc)` returns a vector of **n** values in one go.

The length of the double reference vector, **r**, in `nag_rand_poisson (g05tjc)`, needs to be a different length from the equivalent argument in `nag_ref_vec_poisson (g05ecc)`, see the documentation for more details.

The integer array **state** in the call to `nag_rand_poisson (g05tjc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_poisson (g05tjc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_poisson (g05tjc)` is likely to be different from those produced by a combination of `nag_ref_vec_poisson (g05ecc)` and `nag_return_discrete (g05eyc)`.

nag_ref_vec_binomial (g05edc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_binomial (g05tac)`.

```

Old:
/* nag_ref_vec_binomial (g05edc) */
nag_ref_vec_binomial(m, p, &r, &fail);
for (i = 0; i < n; i++)
    /* nag_return_discrete (g05eyc) */
    x[i] = nag_return_discrete(r);

New:
mode = Nag_InitializeAndGenerate;
lr = 22 + 20 * ((Integer) sqrt(m * p * (1 - p)));
r = NAG_ALLOC(lr, double);

/* nag_rand_binomial (g05tac) */
nag_rand_binomial(mode, n, m, p, r, lr, state, x, &fail);

```

The old function `nag_ref_vec_binomial (g05edc)` sets up a reference vector for use by `nag_return_discrete (g05eyc)`. The replacement function `nag_rand_binomial (g05tac)` is now used to both set up a reference vector and generate the required variates. Setting **mode** = `Nag_InitializeReference` in the call to `nag_rand_binomial (g05tac)` sets up the double reference vector **r** and hence mimics the functionality of `nag_ref_vec_binomial (g05edc)`. Setting **mode** = `Nag_GenerateFromReference` generates a series of variates from a reference vector mimicking the functionality of `nag_return_discrete (g05eyc)` for this particular distribution. Setting **mode** = `Nag_InitializeAndGenerate` initializes the reference vector and generates the variates in one go.

The function `nag_return_discrete (g05eyc)` returns a single variate at a time, whereas the new function `nag_rand_binomial (g05tac)` returns a vector of **n** values in one go.

The length of the double reference vector, **r**, in `nag_rand_binomial (g05tac)`, needs to be a different length from the equivalent argument in `nag_ref_vec_binomial (g05edc)`, see the documentation for more details.

The integer array **state** in the call to `nag_rand_binomial (g05tac)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_binomial (g05tac)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The

required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_binomial (g05tac)` is likely to be different from those produced by a combination of `nag_ref_vec_binomial (g05edc)` and `nag_return_discrete (g05eyc)`.

nag_ran_permut_vec (g05ehc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_permute (g05ncc)`.

```
Old:
    /* nag_ran_permut_vec (g05ehc) */
    nag_ran_permut_vec(index,n,&fail);
New:
    /* nag_rand_permute (g05ncc) */
    nag_rand_permute(index,n,state,&fail);
```

The integer array **state** in the call to `nag_rand_permute (g05ncc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_permute (g05ncc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_permute (g05ncc)` is likely to be different from those produced by `nag_ran_permut_vec (g05ehc)`.

nag_ran_sample_vec (g05ejc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_sample (g05ndc)`.

```
Old:
    /* nag_ran_sample_vec (g05ejc) */
    nag_ran_sample_vec(ia,n,iz,m,&fail);
New:
    /* nag_rand_sample (g05ndc) */
    nag_rand_sample(ia,n,iz,m,state,&fail);
```

The integer array **state** in the call to `nag_rand_sample (g05ndc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_sample (g05ndc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_sample (g05ndc)` is likely to be different from those produced by `nag_ran_sample_vec (g05ejc)`.

nag_ref_vec_discrete_pdf_cdf (g05exc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_gen_discrete (g05tdc)`.

```
Old:
    /* nag_ref_vec_discrete_pdf_cdf (g05exc) */
    nag_ref_vec_discrete_pdf_cdf(p,np,sizep,distf,&r,&fail);
    for (i = 0; i < n; i++)
        /* nag_return_discrete (g05eyc) */
        x[i] = nag_return_discrete(r);
New:
    mode = Nag_InitializeAndGenerate;
    lr = 10 + (Integer) (1.4 * np);
    r = NAG_ALLOC(lr,double);

    /* nag_rand_gen_discrete (g05tdc) */
    nag_rand_gen_discrete(mode,n,p,np,sizep,distf,r,lr,state,x,&fail);
```

The old function `nag_ref_vec_discrete_pdf_cdf (g05exc)` sets up a reference vector for use by `nag_return_discrete (g05eyc)`. The replacement function `nag_rand_gen_discrete (g05tdc)` is now used to both set up a reference vector and generate the required variates. Setting **mode** = `Nag_InitializeReference` in the call to `nag_rand_gen_discrete (g05tdc)` sets up the double reference vector **r** and hence mimics the functionality of `nag_ref_vec_discrete_pdf_cdf (g05exc)`. Setting **mode** = `Nag_GenerateFromReference` generates a series of variates from a reference vector mimicking the functionality of `nag_return_discrete`

(g05eyc) for this particular distribution. Setting `mode = Nag_InitializeAndGenerate` initializes the reference vector and generates the variates in one go.

The function `nag_return_discrete (g05eyc)` returns a single variate at a time, whereas the new function `nag_rand_gen_discrete (g05tdc)` returns a vector of `n` values in one go.

The length of the double reference vector, `r`, in `nag_rand_gen_discrete (g05tdc)`, needs to be a different length from the equivalent argument in `nag_ref_vec_discrete_pdf_cdf (g05exc)`, see the documentation for more details.

The integer array `state` in the call to `nag_rand_gen_discrete (g05tdc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_gen_discrete (g05tdc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array `state` will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_gen_discrete (g05tdc)` is likely to be different from those produced by a combination of `nag_ref_vec_discrete_pdf_cdf (g05exc)` and `nag_return_discrete (g05eyc)`.

nag_return_discrete (g05eyc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_gen_discrete (g05tdc)`.

There is no direct replacement function for `nag_return_discrete (g05eyc)`.

`nag_return_discrete (g05eyc)` is designed to generate random draws from a distribution defined by a reference vector. These reference vectors are created by other functions in Chapter g05, for example `nag_ref_vec_poisson (g05ecc)`, which have themselves been superseded. In order to replace a call to `nag_return_discrete (g05eyc)` you must identify which NAG function generated the reference vector being used and look up its replacement. For example, to replace a call to `nag_return_discrete (g05eyc)` preceded by a call to `nag_ref_vec_discrete_pdf_cdf (g05exc)`, as in:

```
/* nag_ref_vec_discrete_pdf_cdf (g05exc) */
nag_ref_vec_discrete_pdf_cdf(p,np,sizep,distf,&r,&fail);
/* nag_return_discrete (g05eyc) */
x = nag_return_discrete(r);
```

you would need to look at the replacement function for `nag_ref_vec_discrete_pdf_cdf (g05exc)`.

nag_return_multi_normal (g05ezc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_matrix_multi_normal (g05rzc)`.

Old:

```
#define X(I,J) x[(I*pdx + J)]
/* nag_ref_vec_multi_normal (g05eac) */
nag_ref_vec_multi_normal(a,m,c,tdc,eps,&r,&fail);
for (i = 0; i < n; i++) {
    /* nag_return_multi_normal (g05ezc) */
    nag_return_multi_normal(z,r);
    for (j = 0; j < m; j++)
        X(i,j) = z[j];
}
```

New:

```
order = Nag_RowMajor;
mode = Nag_InitializeAndGenerate;
lr = m * (m + 1) + 1;      r = NAG_ALLOC(lr,double);
/* nag_rand_matrix_multi_normal (g05rzc) */
nag_rand_matrix_multi_normal(order,mode,n,m,a,c,tdc,r,lr,
    state,x,pdx,&fail);
```

The old function `nag_ref_vec_multi_normal (g05eac)` sets up a reference vector for use by `nag_return_multi_normal (g05ezc)`. The functionality of both these functions has been combined into the single new function `nag_rand_matrix_multi_normal (g05rzc)`. Setting

mode = Nag_InitializeAndGenerate in the call to nag_rand_matrix_multi_normal (g05rzc) sets up the double reference vector **r** and generates the draws from the multivariate Normal distribution in one go.

The old function nag_return_multi_normal (g05ezc) returns a single (**m**-dimensional vector) draw from the multivariate Normal distribution at a time, whereas the new function nag_rand_matrix_multi_normal (g05rzc) returns an **n** by **m** matrix of **n** draws in one go.

The integer array **state** in the call to nag_rand_matrix_multi_normal (g05rzc) contains information on the base generator being used. This array must have been initialized prior to calling nag_rand_matrix_multi_normal (g05rzc) with a call to either nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc). The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by nag_rand_matrix_multi_normal (g05rzc) is likely to be different from those produced by nag_return_multi_normal (g05ezc).

nag_random_beta (g05fec)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_beta (g05sbc).

```
Old:
    /* nag_random_beta (g05fec) */
    nag_random_beta(a,b,n,x,&fail);
New:
    /* nag_rand_beta (g05sbc) */
    nag_rand_beta(n,a,b,state,x,&fail);
```

The integer array **state** in the call to nag_rand_beta (g05sbc) contains information on the base generator being used. This array must have been initialized prior to calling nag_rand_beta (g05sbc) with a call to either nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc). The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by nag_rand_beta (g05sbc) is likely to be different from those produced by nag_random_beta (g05fec).

nag_random_gamma (g05ffc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_gamma (g05sjc).

```
Old:
    /* nag_random_gamma (g05ffc) */
    nag_random_gamma(a,b,n,x,&fail);
New:
    /* nag_rand_gamma (g05sjc) */
    nag_rand_gamma(n,a,b,state,x,&fail);
```

The integer array **state** in the call to nag_rand_gamma (g05sjc) contains information on the base generator being used. This array must have been initialized prior to calling nag_rand_gamma (g05sjc) with a call to either nag_rand_init_repeatable (g05kfc) or nag_rand_init_nonrepeatable (g05kgc). The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by nag_rand_gamma (g05sjc) is likely to be different from those produced by nag_random_gamma (g05ffc).

nag_arma_time_series (g05hac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_arma (g05phc).

```
Old:
    /* nag_arma_time_series (g05hac) */
    nag_arma_time_series(start,p,q,phi,theta,mean,vara,n,w,ref,&fail);
New:
    mode = (start == Nag_TRUE) ? Nag_InitializeAndGenerate : Nag_Generate-
FromReference;
    lr = (p > q + 1) ? p : q + 1;
    lr += p + q + 6;
    r = NAG_ALLOC(lr,double);
```

```

/* nag_rand_arma (g05phc) */
nag_rand_arma(mode,n,mean,p,phi,q,theta,vara,r,lr,state,&var,x,&fail);

```

The integer array **state** in the call to `nag_rand_arma (g05phc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_arma (g05phc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_arma (g05phc)` is likely to be different from those produced by `nag_arma_time_series (g05hac)`.

nag_generate_agarchI (g05hkc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_agarchI (g05pdc)`.

```

Old:
/* nag_generate_agarchI (g05hkc) */
nag_generate_agarchI(num,p,q,theta,gamma,ht,et,fcall,rvec,&fail);
New:
dist = Nag_NormalDistn;
df = 0;
bfcall = (fcall == Nag_Garch_Fcall_True) ? Nag_TRUE : Nag_FALSE;
lr = 2 * (p + q + 2);
r = NAG_ALLOC(lr,double);

/* nag_rand_agarchI (g05pdc) */
nag_rand_agarchI(dist,num,p,q,theta,gamma,df,ht,et,bfcall,r,lr,
state,&fail);

```

The integer array **state** in the call to `nag_rand_agarchI (g05pdc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_agarchI (g05pdc)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_agarchI (g05pdc)` is likely to be different from those produced by `nag_generate_agarchI (g05hkc)`.

nag_generate_agarchII (g05hlc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_agarchII (g05pec)`.

```

Old:
/* nag_generate_agarchII (g05hlc) */
nag_generate_agarchII(num,p,q,theta,gamma,ht,et,fcall,rvec,&fail);
New:
dist = Nag_NormalDistn;
df = 0;
bfcall = (fcall == Nag_Garch_Fcall_True) ? Nag_TRUE : Nag_FALSE;
lr = 2 * (p + q + 2);
r = NAG_ALLOC(lr,double);

/* nag_rand_agarchII (g05pec) */
nag_rand_agarchII(dist,num,p,q,theta,gamma,df,ht,et,bfcall,r,lr,
state,&fail);

```

The integer array **state** in the call to `nag_rand_agarchII (g05pec)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_agarchII (g05pec)` with a call to either `nag_rand_init_repeatable (g05kfc)` or `nag_rand_init_nonrepeatable (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_agarchII (g05pec)` is likely to be different from those produced by `nag_generate_agarchII (g05hlc)`.

nag_generate_garchGJR (g05hmc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_garchGJR (g05pfc)`.

```

Old:
/* nag_generate_garchGJR (g05hmc) */

```

```

nag_generate_garchGJR(num,p,q,theta,gamma,ht,et,fcall,rvec,&fail);
New:
dist = Nag_NormalDistn;
df = 0;
bfcall = (fcall == Nag_Garch_Fcall_True) ? Nag_TRUE : Nag_FALSE;
lr = 2 * (p + q + 2);
r = NAG_ALLOC(lr,double);

/* nag_rand_garchGJR (g05pfc) */
nag_rand_garchGJR(dist,num,p,q,theta,gamma,df,ht,et,bfcall,r,lr,
state,&fail);

```

The integer array **state** in the call to `nag_rand_garchGJR (g05pfc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_garchGJR (g05pfc)` with a call to either `nag_rand_init_repeatabe (g05kfc)` or `nag_rand_init_nonrepeatabe (g05kgc)`. The required length of the array **state** will depend on the base generator chosen during initialization. Due to changes in the underlying code the sequence of values produced by `nag_rand_garchGJR (g05pfc)` is likely to be different from those produced by `nag_generate_garchGJR (g05hmc)`.

nag_rngs_basic (g05kac)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_basic (g05sac)`.

```

Old:
for (i = 0; i < n; i++)
/* nag_rngs_basic (g05kac) */
x[i] = nag_rngs_basic(igen,iseed);
New:
/* nag_rand_basic (g05sac) */
nag_rand_basic(n,state,x,&fail);

```

nag_rngs_init_repeatabe (g05kbc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_init_repeatabe (g05kfc)`.

```

Old:
/* nag_rngs_init_repeatabe (g05kbc) */
nag_rngs_init_repeatabe(&igen,iseed);
New:
if (igen == 0) {
genid = Nag_Basic;
subid = 1;
} else if (igen >= 1) {
genid = Nag_WichmannHill_I;
subid = igen;
}

/* nag_rand_init_repeatabe (g05kfc) */
nag_rand_init_repeatabe(genid,subid,iseed,lseed,state,&lstate,&fail);

```

nag_rngs_init_nonrepeatabe (g05kcc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_rand_init_nonrepeatabe (g05kgc)`.

```

Old:
/* nag_rngs_init_nonrepeatabe (g05kcc) */
nag_rngs_init_nonrepeatabe(&igen,iseed);
New:
if (igen == 0) {
genid = Nag_Basic;
subid = 1;
} else if (igen >= 1) {
genid = Nag_WichmannHill_I;
subid = igen;
}

/* nag_rand_init_nonrepeatabe (g05kgc) */

```

```
nag_rand_init_nonrepeatable(genid,subid,state,&lstate,&fail);
```

nag_rngs_logical (g05kec)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_logical (g05tbc).

```
Old:
    for (i = 0; i < n; i++)
        /* nag_rngs_logical (g05kec) */
        x[i] = nag_rngs_logical(p,igen,iseed,&fail);
New:
    /* nag_rand_logical (g05tbc) */
    nag_rand_logical(n,p,state,x,&fail);
```

nag_rngs_normal (g05lac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_normal (g05skc).

```
Old:
    /* nag_rngs_normal (g05lac) */
    nag_rngs_normal(xmu,var,n,x,igen,iseed,&fail);
New:
    /* nag_rand_normal (g05skc) */
    nag_rand_normal(n,xmu,var,state,x,&fail);
```

nag_rngs_students_t (g05lbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_students_t (g05snc).

```
Old:
    /* nag_rngs_students_t (g05lbc) */
    nag_rngs_students_t(df,n,x,igen,iseed,&fail);
New:
    /* nag_rand_students_t (g05snc) */
    nag_rand_students_t(n,df,state,x,&fail);
```

nag_rngs_chi_sq (g05lcc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_chi_sq (g05sdc).

```
Old:
    /* nag_rngs_chi_sq (g05lcc) */
    nag_rngs_chi_sq(df,n,x,igen,iseed,&fail);
New:
    /* nag_rand_chi_sq (g05sdc) */
    nag_rand_chi_sq(n,df,state,x,&fail);
```

nag_rngs_f (g05ldc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_f (g05shc).

```
Old:
    /* nag_rngs_f (g05ldc) */
    nag_rngs_f(df1,df2,n,x,igen,iseed,&fail);
New:
    /* nag_rand_f (g05shc) */
    nag_rand_f(n,df1,df2,state,x,&fail);
```

nag_rngs_beta (g05lec)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_beta (g05sbc).

```
Old:
    /* nag_rngs_beta (g05lec) */
    nag_rngs_beta(a,b,n,x,igen,iseed,&fail);
```

New:

```
/* nag_rand_beta (g05sbc) */
nag_rand_beta(n,a,b,state,x,&fail);
```

nag_rngs_gamma (g05lfc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_gamma (g05sjc).

Old:

```
/* nag_rngs_gamma (g05lfc) */
nag_rngs_gamma(a,b,n,x,igen,iseed,&fail);
```

New:

```
/* nag_rand_gamma (g05sjc) */
nag_rand_gamma(n,a,b,state,x,&fail);
```

nag_rngs_uniform (g05lgc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_uniform (g05sqc).

Old:

```
/* nag_rngs_uniform (g05lgc) */
nag_rngs_uniform(a,b,n,x,igen,iseed,&fail);
```

New:

```
/* nag_rand_uniform (g05sqc) */
nag_rand_uniform(n,a,b,state,x,&fail);
```

nag_rngs_triangular (g05lhc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_triangular (g05spc).

Old:

```
/* nag_rngs_triangular (g05lhc) */
nag_rngs_triangular(xmin,xmax,xmed,n,x,igen,iseed,&fail);
```

New:

```
/* nag_rand_triangular (g05spc) */
nag_rand_triangular(n,xmin,xmed,xmax,state,x,&fail);
```

nag_rngs_exp (g05ljc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_exp (g05sfc).

Old:

```
/* nag_rngs_exp (g05ljc) */
nag_rngs_exp(a,n,x,igen,iseed,&fail);
```

New:

```
/* nag_rand_exp (g05sfc) */
nag_rand_exp(n,a,state,x,&fail);
```

nag_rngs_lognormal (g05lkc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_lognormal (g05smc).

Old:

```
/* nag_rngs_lognormal (g05lkc) */
nag_rngs_lognormal(xmu,var,n,x,igen,iseed,&fail);
```

New:

```
/* nag_rand_lognormal (g05smc) */
nag_rand_lognormal(n,xmu,var,state,x,&fail);
```

nag_rngs_cauchy (g05llc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_cauchy (g05scc).

Old:

```
/* nag_rngs_cauchy (g05llc) */
nag_rngs_cauchy(xmed,semiqr,n,x,igen,iseed,&fail);
```

```
New:
    /* nag_rand_cauchy (g05scc) */
    nag_rand_cauchy(n,xmed,semiqr,state,x,&fail);
```

nag_rngs_weibull (g05lmc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_weibull (g05ssc).

```
Old:
    /* nag_rngs_weibull (g05lmc) */
    nag_rngs_weibull(a,b,n,x,igen,iseed,&fail);
New:
    /* nag_rand_weibull (g05ssc) */
    nag_rand_weibull(n,a,b,state,x,&fail);
```

nag_rngs_logistic (g05lnc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_logistic (g05slc).

```
Old:
    /* nag_rngs_logistic (g05lnc) */
    nag_rngs_logistic(a,b,n,x,igen,iseed,&fail);
New:
    /* nag_rand_logistic (g05slc) */
    nag_rand_logistic(n,a,b,state,x,&fail);
```

nag_rngs_von_mises (g05lpc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_von_mises (g05src).

```
Old:
    /* nag_rngs_von_mises (g05lpc) */
    nag_rngs_von_mises(vk,n,x,igen,iseed,&fail);
New:
    /* nag_rand_von_mises (g05src) */
    nag_rand_von_mises(n,vk,state,x,&fail);
```

nag_rngs_exp_mix (g05lqc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_exp_mix (g05sgc).

```
Old:
    /* nag_rngs_exp_mix (g05lqc) */
    nag_rngs_exp_mix(nmix,a,wgt,n,x,igen,iseed,&fail);
New:
    /* nag_rand_exp_mix (g05sgc) */
    nag_rand_exp_mix(n,nmix,a,wgt,state,x,&fail);
```

nag_rngs_matrix_multi_students_t (g05lxc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_matrix_multi_students_t (g05ryc).

```
Old:
    /* nag_rngs_matrix_multi_students_t (g05lxc) */
    nag_rngs_matrix_multi_students_t(order,mode,df,m,xmu,c,pdc,n,x,pdx,
    igen,iseed,r,lr,&fail);
New:
    if (mode == 0) {
        emode = Nag_InitializeAndGenerate;
    } else if (mode == 1) {
        emode = Nag_InitializeReference;
    } else if (mode == 2) {
        emode = Nag_GenerateFromReference;
    }
    lr = m * (m + 1) + 2;
    r = NAG_ALLOC(lr,double);
```

```
/* nag_rand_matrix_multi_students_t (g05ryc) */
nag_rand_matrix_multi_students_t(order,emode,n,df,m,xmu,c,pdc,r,lr,
state,x,pdx,&fail);
```

nag_rgsn_matrix_multi_normal (g05lyc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_matrix_multi_normal (g05rzc).

Old:

```
/* nag_rgsn_matrix_multi_normal (g05lyc) */
nag_rgsn_matrix_multi_normal(order,mode,m,xmu,c,pdc,n,x,pdx,igen,
iseed,r,lr,&fail);
```

New:

```
if (mode == 0) {
    emode = Nag_InitializeAndGenerate;
} else if (mode == 1) {
    emode = Nag_InitializeReference;
} else if (mode == 2) {
    emode = Nag_GenerateFromReference;
}
lr = m * (m + 1) + 1;
r = NAG_ALLOC(lr,double);

/* nag_rand_matrix_multi_normal (g05rzc) */
nag_rand_matrix_multi_normal(order,emode,n,m,xmu,c,pdc,r,lr,
state,x,pdx,&fail);
```

nag_rngs_multi_normal (g05lzc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_matrix_multi_normal (g05rzc).

Old:

```
/* nag_rngs_multi_normal (g05lzc) */
nag_rngs_multi_normal(order,mode,m,xmu,c,pdc,x,igen,iseed,r,&fail);
```

New:

```
if (mode == 0) {
    emode = Nag_InitializeAndGenerate;
} else if (mode == 1) {
    emode = Nag_InitializeReference;
} else if (mode == 2) {
    emode = Nag_GenerateFromReference;
}
n = 1;
pdx = 1;
lr = m * (m + 1) + 1;
r = NAG_ALLOC(lr,double);

/* nag_rand_matrix_multi_normal (g05rzc) */
nag_rand_matrix_multi_normal(order,emode,n,m,xmu,c,pdc,r,lr,
state,x,pdx,&fail);
```

nag_rngs_discrete_uniform (g05mac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_discrete_uniform (g05tlc).

Old:

```
/* nag_rngs_discrete_uniform (g05mac) */
nag_rngs_discrete_uniform(a,b,n,x,igen,iseed,&fail);
```

New:

```
/* nag_rand_discrete_uniform (g05tlc) */
nag_rand_discrete_uniform(n,a,b,state,x,&fail);
```

nag_rngs_geom (g05mbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_geom (g05tcc).

Old:

```
/* nag_rngs_geom (g05mbc) */
```

```

nag_rngs_geom(mode,p,n,x,igen,iseed,r,&fail);
New:
if (mode == 0) {
    emode = Nag_InitializeReference;
} else if (mode == 1) {
    emode = Nag_GenerateFromReference;
} else if (mode == 2) {
    emode = Nag_InitializeAndGenerate;
} else if (mode == 3) {
    emode = Nag_GenerateWithoutReference;
}
lr = (emode == Nag_GenerateWithoutReference) ? 1 :
      8 + (Integer) (42 / p);
r = NAG_ALLOC(lr,double);

/* nag_rand_geom (g05tcc) */
nag_rand_geom(emode,n,p,r,lr,state,x,&fail);

```

nag_rngs_neg_bin (g05mcc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_neg_bin (g05thc).

```

Old:
/* nag_rngs_neg_bin (g05mcc) */
nag_rngs_neg_bin(mode,m,p,n,x,igen,iseed,r,&fail);
New:
if (mode == 0) {
    emode = Nag_InitializeReference;
} else if (mode == 1) {
    emode = Nag_GenerateFromReference;
} else if (mode == 2) {
    emode = Nag_InitializeAndGenerate;
} else if (mode == 3) {
    emode = Nag_GenerateWithoutReference;
}
lr = (emode == Nag_GenerateWithoutReference) ? 1 :
      28 + (Integer) ((20 * sqrt(m*p) + 30 * p) / (1 - p));
r = NAG_ALLOC(lr,double);

/* nag_rand_neg_bin (g05thc) */
nag_rand_neg_bin(emode,n,m,p,r,lr,state,x,&fail);

```

nag_rngs_logarithmic (g05mdc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_logarithmic (g05tfc).

```

Old:
/* nag_rngs_logarithmic (g05mdc) */
nag_rngs_logarithmic(mode,a,n,x,igen,iseed,r,&fail);
New:
if (mode == 0) {
    emode = Nag_InitializeReference;
} else if (mode == 1) {
    emode = Nag_GenerateFromReference;
} else if (mode == 2) {
    emode = Nag_InitializeAndGenerate;
} else if (mode == 3) {
    emode = Nag_GenerateWithoutReference;
}
lr = (emode == Nag_GenerateWithoutReference) ? 1 :
      18 + (Integer) (40 / (1 - a));
r = NAG_ALLOC(lr,double);

/* nag_rand_logarithmic (g05tfc) */
nag_rand_logarithmic(emode,n,a,r,lr,state,x,&fail);

```

nag_rngs_compound_poisson (g05mec)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_compound_poisson (g05tkc).

```

Old:
    /* nag_rngs_compound_poisson (g05mec) */
    nag_rngs_compound_poisson(m,vlamda,x,igen,iseed,&fail);
New:
    /* nag_rand_compound_poisson (g05tkc) */
    nag_rand_compound_poisson(m,vlamda,state,x,&fail);

```

nag_rngs_binomial (g05mjc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_binomial (g05tac).

```

Old:
    /* nag_rngs_binomial (g05mjc) */
    nag_rngs_binomial(mode,m,p,n,x,igen,iseed,r,&fail);
New:
    if (mode == 0) {
        emode = Nag_InitializeReference;
    } else if (mode == 1) {
        emode = Nag_GenerateFromReference;
    } else if (mode == 2) {
        emode = Nag_InitializeAndGenerate;
    } else if (mode == 3) {
        emode = Nag_GenerateWithoutReference;
    }
    lr = (emode == Nag_GenerateWithoutReference) ? 1 :
        22 + 20 * ((Integer) sqrt(m * p * (1 - p)));
    r = NAG_ALLOC(lr,double);

    /* nag_rand_binomial (g05tac) */
    nag_rand_binomial(emode,n,m,p,r,lr,state,x,&fail);

```

nag_rngs_poisson (g05mkc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_poisson (g05tjc).

```

Old:
    /* nag_rngs_poisson (g05mkc) */
    nag_rngs_poisson(mode,lambda,n,x,igen,iseed,r,&fail);
New:
    if (mode == 0) {
        emode = Nag_InitializeReference;
    } else if (mode == 1) {
        emode = Nag_GenerateFromReference;
    } else if (mode == 2) {
        emode = Nag_InitializeAndGenerate;
    } else if (mode == 3) {
        emode = Nag_GenerateWithoutReference;
    }
    lr = (emode == Nag_GenerateWithoutReference) ? 1 : 30 +
        (Integer) (20 * sqrt(lambda) + lambda);
    r = NAG_ALLOC(lr,double);

    /* nag_rand_poisson (g05tjc) */
    nag_rand_poisson(emode,n,lambda,r,lr,state,x,&fail);

```

nag_rngs_hypergeometric (g05mlc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_hypergeometric (g05tec).

```

Old:
    /* nag_rngs_hypergeometric (g05mlc) */
    nag_rngs_hypergeometric(mode,ns,np,m,n,x,igen,iseed,r,&fail);
New:
    if (mode == 0) {

```

```

    emode = Nag_InitializeReference;
} else if (mode == 1) {
    emode = Nag_GenerateFromReference;
} else if (mode == 2) {
    emode = Nag_InitializeAndGenerate;
} else if (mode == 3) {
    emode = Nag_GenerateWithoutReference;
}
lr = (emode == Nag_GenerateWithoutReference) ? 1 : 28 + 20 *
    ((Integer) sqrt((ns * m * (np - m) * (np - ns)) /
        (np * np * np)));
r = NAG_ALLOC(lr, double);

/* nag_rand_hypergeometric (g05tec) */
nag_rand_hypergeometric(emode, n, ns, np, m, r, lr, state, x, &fail);

```

nag_rngs_gen_multinomial (g05mrc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_gen_multinomial (g05tgc).

```

Old:
/* nag_rngs_gen_multinomial (g05mrc) */
nag_rngs_gen_multinomial(order, mode, m, k, p, n, x, pdx, igen, iseed, r, &fail);
New:
if (mode == 0) {
    emode = Nag_InitializeReference;
} else if (mode == 1) {
    emode = Nag_GenerateFromReference;
} else if (mode == 2) {
    emode = Nag_InitializeAndGenerate;
} else if (mode == 3) {
    emode = Nag_GenerateWithoutReference;
}
pmax = p[0];
for (i = 1; i < k; i++)
    pmax = (pmax > p[i]) ? p[i] : pmax;
lr = (emode == Nag_GenerateWithoutReference) ? 1 : 30 +
    20 * ((Integer) sqrt(m * pmax * (1 - pmax)));
r = NAG_ALLOC(lr, double);

/* nag_rand_gen_multinomial (g05tgc) */
nag_rand_gen_multinomial(order, emode, n, m, k, p, r, lr, state, x, pdx, &fail);

```

nag_rngs_gen_discrete (g05mzc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_gen_discrete (g05tdc).

```

Old:
/* nag_rngs_gen_discrete (g05mzc) */
nag_rngs_gen_discrete(mode, p, np, ip1, comp_type, n, x, igen, iseed, r, &fail);
New:
if (mode == 0) {
    emode = Nag_InitializeReference;
} else if (mode == 1) {
    emode = Nag_GenerateFromReference;
} else if (mode == 2) {
    emode = Nag_InitializeAndGenerate;
}
itype = (comp_type == Nag_Compute_1) ? Nag_PDF : Nag_CDF;
lr = 10 + (Integer) (1.4 * np);
r = NAG_ALLOC(lr, double);

/* nag_rand_gen_discrete (g05tdc) */
nag_rand_gen_discrete(emode, n, p, np, ip1, itype, r, lr, state, x, &fail);

```

nag_rngs_permute (g05nac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_permute (g05ncc).

```

Old:
    /* nag_rngs_permute (g05nac) */
    nag_rngs_permute(index,n,igen,iseed,&fail);
New:
    /* nag_rand_permute (g05ncc) */
    nag_rand_permute(index,n,state,&fail);

```

nag_rngs_sample (g05nbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_sample (g05ndc).

```

Old:
    /* nag_rngs_sample (g05nbc) */
    nag_rngs_sample(ipop,n,isampl,m,igen,iseed,&fail);
New:
    /* nag_rand_sample (g05ndc) */
    nag_rand_sample(ipop,n,isampl,m,state,&fail);

```

nag_rngs_arma_time_series (g05pac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_arma (g05phc).

```

Old:
    /* nag_rngs_arma_time_series (g05pac) */
    nag_rngs_arma_time_series(mode,xmean,p,phi,q,theta,avar,&var,n,x,
        igen,iseed,r,&fail);
New:
    if (mode == 0) {
        emode = Nag_InitializeReference;
    } else if (mode == 1) {
        emode = Nag_GenerateFromReference;
    } else if (mode == 2) {
        emode = Nag_InitializeAndGenerate;
    }
    lr = p + q + 6 * ((p < q + 1) ? q + 1 : p);
    r = NAG_ALLOC(lr,double);

    /* nag_rand_arma (g05phc) */
    nag_rand_arma(emode,n,xmean,p,phi,q,theta,avar,r,lr,state,&var,x,
        &fail);

```

nag_rngs_varma_time_series (g05pcc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_varma (g05pjc).

```

Old:
    /* nag_rngs_varma_time_series (g05pcc) */
    nag_rngs_varma_time_series(order,mode,k,xmean,p,phi,q,theta,
        var,pdv,n,x,pdx,igen,iseed,r,&fail);
New:
    if (mode == 0) {
        emode = Nag_InitializeReference;
    } else if (mode == 1) {
        emode = Nag_GenerateFromReference;
    } else if (mode == 2) {
        emode = Nag_InitializeAndGenerate;
    } else if (mode == 3) {
        emode = Nag_ReGenerateFromReference;
    }
    tmp1 = (p > q) ? p : q;
    if (p == 0) {
        tmp2 = k * (k + 1) / 2;
    } else {
        tmp2 = k*(k+1)/2 + (p-1)*k*k;
    }

```

```

}
tmp3 = p + q;
if (k >= 6) {
  lr = (5*tmp1*tmp1+1)*k*k + (4*tmp1+3)*k + 4;
} else {
  tmp4 = k*tmp1*(k*tmp1+2);
  tmp5 = k*k*tmp3*tmp3+tmp2*(tmp2+3)+k*k*(q+1);
  lr = (tmp3*tmp3+1)*k*k + (4*tmp3+3)*k +
      ((tmp4 > tmp5) ? tmp4 : tmp5) + 4;
}
r = NAG_ALLOC(lr,double);

/* nag_rand_varma (g05pjc) */
nag_rand_varma(order,emode,n,k,xmean,p,phi,q,theta,var,pdv,r,lr,
  state,x,pdx,&fail);

```

nag_rngs_orthog_matrix (g05qac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_orthog_matrix (g05pxc).

```

Old:
/* nag_rngs_orthog_matrix (g05qac) */
nag_rngs_orthog_matrix(order,side,init,m,n,a,pda,igen,iseed,&fail);
New:
if (order == Nag_RowMajor) {
  /* nag_rand_orthog_matrix (g05pxc) */
  nag_rand_orthog_matrix(side,init,m,n,state,a,pda,&fail);
} else {
  tside = (side == Nag_LeftSide) ? Nag_RightSide : Nag_LeftSide;
  pda = m;

  /* nag_rand_orthog_matrix (g05pxc) */
  nag_rand_orthog_matrix(tside,init,n,m,state,a,pda,&fail);
}

```

nag_rngs_corr_matrix (g05qbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_corr_matrix (g05pyc).

```

Old:
/* nag_rngs_corr_matrix (g05qbc) */
nag_rngs_corr_matrix(order,n,d,c,pdc,eps,igen,iseed,&fail);
New:
/* nag_rand_corr_matrix (g05pyc) */
nag_rand_corr_matrix(n,d,eps,state,c,pdc,&fail);

```

nag_rngs_2_way_table (g05qdc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_2_way_table (g05pzc).

```

Old:
/* nag_rngs_2_way_table (g05qdc) */
nag_rngs_2_way_table(order,mode,nrow,ncol,totr,totc,x,pdx,igen,
  iseed,r,nr,&fail);
New:
if (mode == 0) {
  emode = Nag_InitializeReference;
} else if (mode == 1) {
  emode = Nag_GenerateFromReference;
} else if (mode == 2) {
  emode = Nag_InitializeAndGenerate;
}
for (i = 0, lr = 5; i < nrow; i++)
  lr += totr[i];
r = NAG_ALLOC(lr,double);
if (order == Nag_RowMajor) {
  /* nag_rand_2_way_table (g05pzc) */
  nag_rand_2_way_table(emode,nrow,ncol,totr,totc,r,lr,state,x,pdx,

```

```

        &fail);
    } else {
        pdx = nrow;

        /* nag_rand_2_way_table (g05pzc) */
        nag_rand_2_way_table(emode,ncol,nrow,totc,totr,r,lr,state,x,pdx,
            &fail);
    }

```

nag_rngs_copula_normal (g05rac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_copula_normal (g05rdc).

```

Old:
    /* nag_rngs_copula_normal (g05rac) */
    nag_rngs_copula_normal(order,mode,m,c,pdc,n,x,pdx,igen,iseed,r,lr,
        &fail);
New:
    if (mode == 1) {
        emode = Nag_InitializeReference;
    } else if (mode == 2) {
        emode = Nag_GenerateFromReference;
    } else if (mode == 0) {
        emode = Nag_InitializeAndGenerate;
    }
    lr = m * (m + 1) + 1;
    r = NAG_ALLOC(lr,double);

    /* nag_rand_copula_normal (g05rdc) */
    nag_rand_copula_normal(order,emode,n,m,c,pdc,r,lr,state,x,pdx,
        &fail);

```

nag_rngs_copula_students_t (g05rbc)

Scheduled for withdrawal at Mark 11.

Replaced by nag_rand_copula_students_t (g05rcc).

```

Old:
    /* nag_rngs_copula_students_t (g05rbc) */
    nag_rngs_copula_students_t(order,mode,df,m,c,pdc,n,x,pdx,igen,
        iseed,r,lr,&fail);
New:
    if (mode == 1) {
        emode = Nag_InitializeReference;
    } else if (mode == 2) {
        emode = Nag_GenerateFromReference;
    } else if (mode == 0) {
        emode = Nag_InitializeAndGenerate;
    }

    /* nag_rand_copula_students_t (g05rcc) */
    nag_rand_copula_students_t(order,emode,n,df,m,c,pdc,r,lr,
        state,x,pdx,&fail);

```

nag_quasi_random_uniform (g05yac)

Scheduled for withdrawal at Mark 11.

Replaced by nag_quasi_init (g05ylc) and nag_quasi_rand_uniform (g05ymc).

```

Old:
    /* nag_quasi_random_uniform (g05yac) */
    nag_quasi_random_uniform(state,seq,iskip,idim,quasi,&gf,&fail);
New:
    liref = (seq == Nag_QuasiRandom_Faure) ? 407 : 32 * idim + 7;
    iref = NAG_ALLOC(liref,Integer);
    seq = (seq == Nag_QuasiRandom_Sobol) ?
        Nag_QuasiRandom_SobolA659 : seq;

    if (state == Nag_QuasiRandom_Init) {
        /* nag_quasi_init (g05ylc) */

```

```

    nag_quasi_init(seq, idim, iref, liref, iskip, &fail);
} else if (state == Nag_QuasiRandom_Cont) {
    n = 1;
    pdquasi = (order == Nag_RowMajor) ? idim : n;

    /* nag_quasi_rand_uniform (g05ymc) */
    nag_quasi_rand_uniform(order, n, quasi, pdquasi, iref, &fail);
}

```

`nag_quasi_random_uniform (g05yac)` has been split into two functions; `nag_quasi_init (g05ylc)` to initialize the quasi-random generators and `nag_quasi_rand_uniform (g05ymc)` to generate the values. `nag_quasi_rand_uniform (g05ymc)` will generate more than one realization at a time. Information is passed between `nag_quasi_init (g05ylc)` and `nag_quasi_rand_uniform (g05ymc)` using the integer vector **iref** rather than the NAG defined structure **gf**. Therefore there is no longer any need to call a function to release memory as **iref** can be "freed" like any C array.

nag_quasi_random_normal (g05ybc)

Scheduled for withdrawal at Mark 11.

Replaced by `nag_quasi_rand_normal (g05yjc)` and `nag_quasi_init (g05ylc)`.

```

Old:
    /* nag_quasi_random_normal (g05ybc) */
    nag_quasi_random_normal(state, seq, lnorm, mean, std, iskip, idim,
        quasi, &gf, &fail);
New:
    liref = (seq == Nag_QuasiRandom_Faure) ? 407 : 32 * idim + 7;
    iref = NAG_ALLOC(liref, Integer);
    seq = (seq == Nag_QuasiRandom_Sobol) ?
        Nag_QuasiRandom_SobolA659 : seq;

    if (state == Nag_QuasiRandom_Init) {
        /* nag_quasi_init (g05ylc) */
        nag_quasi_init(seq, idim, iref, liref, iskip, &fail);
    } else if (state == Nag_QuasiRandom_Cont) {
        n = 1;
        pdquasi = (order == Nag_RowMajor) ? idim : n;

        if (lnorm == Nag_LogNormal) {
            /* nag_quasi_rand_lognormal (g05ykc) */
            nag_quasi_rand_lognormal(order, mean, std, n, quasi, pdquasi, iref,
                &fail);
        } else if (lnorm == Nag_Normal) {
            /* nag_quasi_rand_normal (g05yjc) */
            nag_quasi_rand_normal(order, mean, std, n, quasi, pdquasi, iref, &fail);
        }
    }
}

```

`nag_quasi_random_normal (g05ybc)` has been split into three functions; `nag_quasi_init (g05ylc)` to initialize the quasi-random generators, `nag_quasi_rand_lognormal (g05ykc)` to generate values from a log-normal distribution and `nag_quasi_rand_normal (g05yjc)` to generate values from a normal distribution. Both `nag_quasi_rand_lognormal (g05ykc)` and `nag_quasi_rand_normal (g05yjc)` will generate more than one realization at a time. Information is passed between `nag_quasi_init (g05ylc)` and `nag_quasi_rand_lognormal (g05ykc)` and `nag_quasi_rand_normal (g05yjc)` using the integer vector **iref** rather than the NAG defined structure **gf**. Therefore there is no longer any need to call a function to release memory as **iref** can be "freed" like any C array.