

## NAG Library

### Advice on Replacement Calls for Withdrawn/Superseded Functions

The following list gives the names of functions that are suitable replacements for functions that have either been withdrawn or superseded since Mark 23.

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.

#### c05 – Roots of One or More Transcendental Equations

##### **nag\_zero\_cont\_func\_bd (c05adc)**

Withdrawn at Mark 24.

Replaced by `nag_zero_cont_func_brent (c05ayc)`.

```
Old: double f(double xx)
    {
        ...
    }
    ...
    nag_zero_cont_func_bd(a, b, &x, f, xtol, ftol, &fail);
New: double f(double xx, Nag_Comm *comm)
    {
        ...
    }
    ...
    Nag_Comm comm;
    ...
    nag_zero_cont_func_brent(a, b, xtol, ftol, f, &x, &comm, &fail);
```

##### **nag\_zero\_cont\_func\_brent\_bsrch (c05agc)**

Withdrawn at Mark 25.

Replaced by `nag_zero_cont_func_brent_binsrch (c05auc)`.

```
Old: nag_zero_cont_func_brent_bsrch(...);
New: nag_zero_cont_func_brent_binsrch(...);
```

##### **nag\_zero\_nonlin\_eqns (c05nbc)**

Withdrawn at Mark 24.

Replaced by `nag_zero_nonlin_eqns_easy (c05qbc)`.

```
Old: void f(Integer n, const double x[], double fvec[], Integer *userflag)
    {
        ...
    }
    ...
    nag_zero_nonlin_eqns(n, x, fvec, f, xtol, &fail);
New: void fcn(Integer n, const double x[], double fvec[], Nag_Comm *comm,
             Integer *userflag)
    {
        ...
    }
    ...
    Nag_Comm comm;
    ...
    nag_zero_nonlin_eqns_easy(fcn, n, x, fvec, xtol, &comm, &fail);
```

**nag\_zero\_nonlin\_eqns\_deriv (c05pbc)**

Withdrawn at Mark 24.

Replaced by nag\_zero\_nonlin\_eqns\_deriv\_easy (c05rbc).

```
Old: void f(Integer n, double x[], double fvec[], double fjac[],
           Integer tdfjac, Integer *userflag)
    {
        ...
    }
    ...
    fjac = NAG_ALLOC(n*tdfjac, double);
    ...
    nag_zero_nonlin_eqns_deriv(n, x, fvec, fjac, tdfjac, f, xtol, &fail);
New: void fcn(Integer n, double x[], double fvec[], double fjac[],
             Nag_Comm *comm, Integer *iflag)
    {
        ...
    }
    ...
    Nag_Comm comm;
    ...
    fjac = NAG_ALLOC(n*n, double);
    ...
    nag_zero_nonlin_eqns_deriv_easy(fcn, n, x, fvec, fjac, xtol, &comm,
                                   &fail);
```

**nag\_zero\_cont\_func\_bd\_1 (c05sdc)**

Withdrawn at Mark 25.

Replaced by nag\_zero\_cont\_func\_brent (c05ayc).

```
Old: double f(double x, Nag_User *comm)
    {
        ...
    }
    ...
    Nag_User comm;
    ...
    nag_zero_cont_func_bd_1(a, b, &x, f, xtol, ftol, &comm, &fail);
New: double f(double xx, Nag_Comm *comm)
    {
        ...
    }
    ...
    Nag_Comm comm;
    ...
    nag_zero_cont_func_brent(a, b, xtol, ftol, f, &x, &comm, &fail);
```

Note that the communication structure **comm** is now of type Nag\_Comm (see Section 3.2.1.1 in the Essential Introduction) rather than Nag\_User (see Section 3.2.1.1 in the Essential Introduction).

**nag\_zero\_nonlin\_eqns\_1 (c05tbc)**

Withdrawn at Mark 24.

Replaced by nag\_zero\_nonlin\_eqns\_easy (c05qbc).

```
Old: void f(Integer n, const double x[], double fvec[], Integer *userflag,
           Nag_User *comm)
    {
        ...
    }
    ...
    Nag_User comm;
    ...
    nag_zero_nonlin_eqns_1(n, x, fvec, fcn, xtol, &comm, &fail);
New: void fcn(Integer n, const double x[], double fvec[], Nag_Comm *comm,
             Integer *userflag)
    {
        ...
    }
    ...
    Nag_Comm comm;
    ...
    nag_zero_nonlin_eqns_easy(fcn, n, x, fvec, xtol, &comm, &fail);
```

Note that the communication structure **comm** is now of type Nag\_Comm (see Section 3.2.1.1 in the Essential Introduction) rather than Nag\_User (see Section 3.2.1.1 in the Essential Introduction).

**nag\_zero\_nonlin\_eqns\_deriv\_1 (c05ubc)**

Withdrawn at Mark 25.

Replaced by nag\_zero\_nonlin\_eqns\_deriv\_easy (c05rbc).

```
Old: void f(Integer n, double x[], double fvec[], double fjac[],
           Integer tdfjac, Integer *userflag, Nag_User *comm)
    {
        ...
    }
    ...
    Nag_User comm;
    ...
    fjac = NAG_ALLOC(n*tdfjac, double);
    ...
    nag_zero_nonlin_eqns_deriv_1(n, x, fvec, fjac, tdfjac, f, xtol,
                               &comm, &fail);
New: void fcn(Integer n, double x[], double fvec[], double fjac[],
             Nag_Comm *comm, Integer *userflag)
    {
        ...
    }
    ...
    Nag_Comm comm;
    ...
    fjac = NAG_ALLOC(n*n, double);
    ...
    nag_zero_nonlin_eqns_deriv_easy(fcn, n, x, fvec, fjac, xtol, &comm,
                                   &fail);
```

Note that the communication structure **comm** is now of type Nag\_Comm (see Section 3.2.1.1 in the Essential Introduction) rather than Nag\_User (see Section 3.2.1.1 in the Essential Introduction).

**nag\_check\_deriv (c05zbc)**

Withdrawn at Mark 24.

Replaced by nag\_check\_derivs (c05zdc).

```
Old: nag_check_deriv(n, x, fvec, fjac, tdfjac, f, &fail);
New: Integer mode, m;
      double *xp = 0, *fvecp = 0, *err = 0;
      m = n;
      mode = 1;
      nag_check_derivs(mode, m, n, x, fvec, fjac, xp, fvecp, err, &fail);
      /* Set fvec to the function values at the original point x and fvecp
       * to the function values at the update point xp. */
      mode = 2;
      nag_check_derivs(mode, m, n, x, fvec, fjac, xp, fvecp, err, &fail);
      /* Check the contents of err for the measures of correctness of each
       * gradient. */
```

**nag\_check\_deriv\_1 (c05zcc)**

Withdrawn at Mark 24.

Replaced by nag\_check\_derivs (c05zdc).

```
Old: nag_check_deriv_1(n, x, fvec, fjac, tdfjac, f, &comm, &fail);
New: Integer mode, m;
      double *xp = 0, *fvecp = 0, *err = 0;
      m = n;
      mode = 1;
      nag_check_derivs(mode, m, n, x, fvec, fjac, xp, fvecp, err, &fail);
      /* Set fvec to the function values at the original point x and fvecp
       * to the function values at the update point xp. */
      mode = 2;
      nag_check_derivs(mode, m, n, x, fvec, fjac, xp, fvecp, err, &fail);
      /* Check the contents of err for the measures of correctness of each
       * gradient. */
```

**c06 – Fourier Transforms****nag\_fft\_real (c06eac)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_realherm\_1d (c06pac).

```
Old: nag_fft_real(n, x, &fail);
New: nag_sum_fft_realherm_1d(Nag_ForwardTransform, x, n, &fail);
```

where the dimension of the array  $\mathbf{x}$  has been extended from the original  $\mathbf{n}$  to  $\mathbf{n} + 2$ . The output values  $\mathbf{x}$  are stored in a different order with real and imaginary parts stored contiguously. The mapping of output elements is as follows:

$$\begin{aligned} \mathbf{x}[2 \times i - 1] &\leftarrow \mathbf{x}[i - 1], \text{ for } i = 0, 1, \dots, \mathbf{n}/2 \text{ and} \\ \mathbf{x}[2 \times i] &\leftarrow \mathbf{x}[\mathbf{n} - i - 1], \text{ for } i = 1, 2, \dots, (\mathbf{n} + 1)/2. \end{aligned}$$

**nag\_fft\_hermitian (c06ebc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_realherm\_1d (c06pac).

```
Old: nag_fft_hermitian(n, x, &fail);
New: nag_sum_fft_realherm_1d(Nag_BackwardTransform, x, n, &fail);
```

where the dimension of the array  $\mathbf{x}$  has been extended from the original  $\mathbf{n}$  to  $\mathbf{n} + 2$ . The input values of  $\mathbf{x}$  are stored in a different order with real and imaginary parts stored contiguously. Also nag\_sum\_fft\_realherm\_1d (c06pac) performs the inverse transform without the need to first conjugate. If prior conjugation of original array  $\mathbf{x}$  is assumed then the mapping of input elements is:

$$\begin{aligned} \mathbf{x}[2 \times i - 1] &\leftarrow \mathbf{x}[i - 1], \text{ for } i = 0, 1, \dots, \mathbf{n}/2 \text{ and} \\ \mathbf{x}[2 \times i] &\leftarrow \mathbf{x}[\mathbf{n} - i - 1], \text{ for } i = 1, 2, \dots, (\mathbf{n} - 1)/2. \end{aligned}$$

**nag\_fft\_complex (c06ecc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_complex\_1d (c06pcc).

Old: `nag_fft_complex(n, x, y, &fail);`

New: `nag_sum_fft_complex_1d(Nag_ForwardTransform, z, n, &fail);`

where  $z$  is a complex array of length  $n$  such that  $z[i].re = x[i]$  and  $z[i].im = y[i]$ , for  $i = 0, 1, \dots, n - 1$  on input and output.

**nag\_convolution\_real (c06ekc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_convcorr\_real (c06fkc).

Old: `nag_convolution_real(job, n, x, y, &fail);`

New: `nag_sum_convcorr_real(job, x, y, n, &fail);`

**nag\_fft\_multiple\_complex (c06frc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_complex\_1d\_multi (c06psc).

Old: `nag_fft_multiple_complex(m, n, x, y, trig, &fail);`

New: `nag_sum_fft_complex_1d_multi(Nag_ForwardTransform, n, m, z, &fail);`

where  $z$  is a complex array of length  $m \times n$  such that  $z[i].re = x[i]$  and  $z[i].im = y[i]$ , for  $i = 0, 1, \dots, m \times n - 1$  on input and output.

**nag\_fft\_2d\_complex (c06fuc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_complex\_2d (c06puc).

Old: `nag_fft_2d_complex(m, n, x, y, trigm, trign, &fail);`

New: `nag_sum_fft_complex_2d(Nag_ForwardTransform, m, n, z, &fail);`

where  $z$  is a complex array of length  $m \times n$  such that  $z[i].re = x[i]$  and  $z[i].im = y[i]$ , for  $i = 0, 1, \dots, m \times n - 1$  on input and output.

**nag\_conjugate\_hermitian (c06gbc)**

Scheduled for withdrawal at Mark 26.

There is no replacement for this function.

**nag\_conjugate\_complex (c06gcc)**

Scheduled for withdrawal at Mark 26.

There is no replacement for this function.

**nag\_fft\_multiple\_sine (c06hac)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_sine (c06rec).

Old: `nag_fft_multiple_sine(m, n, x, trig, &fail);`

New: `nag_sum_fft_sine(m, n, x, &fail);`

**nag\_fft\_multiple\_cosine (c06hbc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_cosine (c06rfc).

Old: `nag_fft_multiple_cosine(m, n, x, trig, &fail);`

New: `nag_sum_fft_cosine(m, n, x, &fail);`

**nag\_fft\_multiple\_qtr\_sine (c06hcc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_qtrsine (c06rgc).

```
Old: nag_fft_multiple_qtr_sine(direct, m, n, x, trig, &fail);  
New: nag_sum_fft_qtrsine(direct, m, n, x, &fail);
```

**nag\_fft\_multiple\_qtr\_cosine (c06hdc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_sum\_fft\_qtrcosine (c06rhc).

```
Old: nag_fft_multiple_qtr_cosine(direct, m, n, x, trig, &fail);  
New: nag_sum_fft_qtrcosine(direct, m, n, x, &fail);
```

**d01 – Quadrature****nag\_1d\_quad\_gen (d01ajc)**

Withdrawn at Mark 24.

Replaced by nag\_1d\_quad\_gen\_1 (d01sjc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **f**.

**nag\_1d\_quad\_osc (d01akc)**

Withdrawn at Mark 24.

Replaced by nag\_1d\_quad\_osc\_1 (d01skc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **f**.

**nag\_1d\_quad\_brkpts (d01alc)**

Withdrawn at Mark 24.

Replaced by nag\_1d\_quad\_brkpts\_1 (d01slc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **f**.

**nag\_1d\_quad\_inf (d01amc)**

Withdrawn at Mark 24.

Replaced by nag\_1d\_quad\_inf\_1 (d01smc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **f**.

**nag\_1d\_quad\_wt\_trig (d01anc)**

Withdrawn at Mark 24.

Replaced by nag\_1d\_quad\_wt\_trig\_1 (d01snc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **g**.

**nag\_1d\_quad\_wt\_alglog (d01apc)**

Withdrawn at Mark 24.

Replaced by nag\_1d\_quad\_wt\_alglog\_1 (d01spc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **g**.

**nag\_1d\_quad\_wt\_cauchy (d01aqc)**

Withdrawn at Mark 24.

Replaced by nag\_1d\_quad\_wt\_cauchy\_1 (d01sqc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **g**.

**nag\_1d\_quad\_inf\_wt\_trig (d01asc)**

Withdrawn at Mark 24.

Replaced by nag\_1d\_quad\_inf\_wt\_trig\_1 (d01ssc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **g**.

**nag\_1d\_quad\_gauss (d01bac)**

Withdrawn at Mark 24.

Replaced by nag\_quad\_1d\_gauss\_vec (d01uac).

```
Old: double fun (double x)
      dinest = nag_1d_quad_gauss(quadrule, fun, a, b, n, &fail);
New: void f (const double x[], Integer nx, double fv[], Integer *iflag,
            Nag_Comm *comm)
      nag_quad_1d_gauss_vec(quadrule, a, b, n, f, &dinest, &comm,
                           &fail);
```

Replace quadrule with quad\_type as follows:

```
Nag_Legendre with Nag_Quad_Gauss_Legendre;
Nag_Rational with Nag_Quad_Gauss_Rational_Adjusted;
Nag_Laguerre with Nag_Quad_Gauss_Laguerre;
Nag_Hermite with Nag_Quad_Gauss_Hermite.
```

**comm** is a pointer to a structure of type Nag\_Comm available to allow you to pass information to the user-supplied function **f**.

**iflag** is an integer which you may use to force an immediate exit from nag\_quad\_1d\_gauss\_vec (d01uac) in case of an error in the user-supplied function **f**.

**f** may be used to call the original **fun** as follows, although it may be more efficient to recode the integrand.

```
void f(const double x[], const Integer nx, double fv[], Integer *iflag,
      Nag_Comm *comm)
{
    Integer i;
    for(i=0; i<nx; i++)
    {
        fv[i] = fun(x[i]);
    }
}
```

**nag\_multid\_quad\_adapt (d01fcc)**

Withdrawn at Mark 25.

Replaced by nag\_multid\_quad\_adapt\_1 (d01wcc).

Where **comm**, a pointer to a structure of type Nag\_User, has been added to allow you to pass information to the user-supplied function **f**.

**nag\_multid\_quad\_monte\_carlo (d01gbc)**

Withdrawn at Mark 25.

Replaced by nag\_multid\_quad\_monte\_carlo\_1 (d01xbc).

Where **comm**, a pointer to a structure of type `Nag_User`, has been added to allow you to pass information to the user-supplied function **f**.

### **nag\_1d\_quad\_gauss\_1 (d01tac)**

Scheduled for withdrawal at Mark 27.

Replaced by `nag_quad_1d_gauss_vec (d01uac)`.

## **d02 – Ordinary Differential Equations**

### **nag\_ode\_ivp\_rk\_range (d02pcc)**

Scheduled for withdrawal at Mark 26.

Replaced by `nag_ode_ivp_rkts_range (d02pec)` and associated `d02p` functions.

Old: `nag_ode_ivp_rk_setup(n, tstart, yinit, tend, tol, thres, method, task, errass, hstart, &opt, &fail);`

```
...
nag_ode_ivp_rk_range(n, f, twant, &tgot, ygot, ypgot, ymax, &opt,
                    &comm,
                    &fail);
```

New: `nag_ode_ivp_rkts_setup(n, tstart, tend, yinit, tol, thres, method, errass, hstart, iwsav, rwsav, &fail);`

```
...
nag_ode_ivp_rkts_range(f2, n, twant, &tgot, ygot, ypgot, ymax,
                      &comm2, iwsav, rwsav, &fail);
```

**iwsav** is an Integer array of length 130 and **rwsav** is a double array of length  $350 + 32 \times n$ .

**comm2** is a pointer to a structure of type `Nag_Comm` available to allow you to pass information to the user defined function **f2** (see **f** in `nag_ode_ivp_rkts_range (d02pec)`).

The definition of **f2** (see **f** in `nag_ode_ivp_rkts_range (d02pec)`) can use the original function **f** as follows:

```
void f2(double t, Integer n, const double *y, double *yp, Nag_Comm *comm2)
{
    Nag_User comm;
    f(n, t, y, yp, &comm);
}
```

### **nag\_ode\_ivp\_rk\_onestep (d02pdc)**

Scheduled for withdrawal at Mark 26.

Replaced by `nag_ode_ivp_rkts_onestep (d02pfc)` and associated `d02p` functions.

Old: `nag_ode_ivp_rk_setup(n, tstart, yinit, tend, tol, thres, method, task, errass, hstart, &opt, &fail);`

```
nag_ode_ivp_rk_onestep(n, f, &tnow, ynow, ypnw, &opt, &comm,
                      &fail);
```

New: `nag_ode_ivp_rkts_setup(n, tstart, tend, yinit, tol, thres, method, errass, hstart, iwsav, rwsav, &fail);`

```
nag_ode_ivp_rkts_onestep(f2, n, &tnow, ynow, ypnw, &comm2, iwsav,
                          rwsav, &fail);
```

**iwsav** is an Integer array of length 130 and **rwsav** is a double array of length  $350 + 32 \times n$ .

**comm2** is a pointer to a structure of type `Nag_Comm` available to allow you to pass information to the user defined function **f2** (see **f** in `nag_ode_ivp_rkts_range (d02pec)`).

The definition of **f2** (see **f** in `nag_ode_ivp_rkts_range (d02pec)`) can use the original function **f** as follows:

```
void f2(double t, Integer n, const double *y, double *yp, Nag_Comm *comm2)
{
    Nag_User comm;
    f(n, t, y, yp, &comm);
}
```



**nag\_ode\_ivp\_rk\_free (d02ppc)**

Scheduled for withdrawal at Mark 26.  
There is no replacement for this function.

**nag\_ode\_ivp\_rk\_setup (d02pvc)**

Scheduled for withdrawal at Mark 26.  
Replaced by nag\_ode\_ivp\_rkts\_setup (d02pqc).

See nag\_ode\_ivp\_rk\_range (d02pcc) and nag\_ode\_ivp\_rk\_onestep (d02pdc) for further information.

**nag\_ode\_ivp\_rk\_reset\_tend (d02pwc)**

Scheduled for withdrawal at Mark 26.  
Replaced by nag\_ode\_ivp\_rkts\_reset\_tend (d02prc).

Old: nag\_ode\_ivp\_rk\_reset\_tend(tendnu, &opt, &fail);  
New: nag\_ode\_ivp\_rkts\_reset\_tend(tendnu, iwsav, rwsav, &fail);

**iwsav** is an Integer array of length 130 and **rwsav** is a double array of length 350.

**nag\_ode\_ivp\_rk\_interp (d02pxc)**

Scheduled for withdrawal at Mark 26.  
Replaced by nag\_ode\_ivp\_rkts\_interp (d02psc).

Old: nag\_ode\_ivp\_rk\_interp(n, twant, request, nwant, ywant, ypwant, f, &opt, &comm, &fail);  
New: nag\_ode\_ivp\_rkts\_interp(n, twant, request, nwant, ywant, ypwant, f2, wcomm, lwcomm, &comm2, iwsav, rwsav, &fail);

**iwsav** is an Integer array of length 130 and **rwsav** is a double array of length  $350 + 32 \times n$ .

**comm2** is a pointer to a structure of type Nag\_Comm available to allow you to pass information to the user defined function f2 (see **f** in nag\_ode\_ivp\_rkts\_interp (d02psc)).

**wcomm** is a double array of length **lwcomm**. See the function document for nag\_ode\_ivp\_rkts\_interp (d02psc) for further information.

The definition of f2 (see **f** in nag\_ode\_ivp\_rkts\_interp (d02psc)) can use the original function **f** as follows:

```
void f2(double t, Integer n, const double *y, double *yp, Nag_Comm *comm2)
{
    Nag_User comm;
    f(n, t, y, yp, &comm);
}
```

**nag\_ode\_ivp\_rk\_errass (d02pzc)**

Scheduled for withdrawal at Mark 26.  
Replaced by nag\_ode\_ivp\_rkts\_errass (d02puc).

Old: nag\_ode\_ivp\_rk\_errass(n, rmserr, &errmax, &terrmx, &opt, &fail);  
New: nag\_ode\_ivp\_rkts\_errass(n, rmserr, &errmax, &terrmx, iwsav, rwsav, &fail);

**n** must be unchanged from that passed to nag\_ode\_ivp\_rkts\_setup (d02pqc).

**iwsav** is an Integer array of length 130 and **rwsav** is a double array of length  $350 + 32 \times n$ .

**e01 – Interpolation****nag\_2d\_scatter\_interpolant (e01sac)**

Withdrawn at Mark 23.  
Replaced by nag\_2d\_shep\_interp (e01sge) 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/numeric/cl/nagdoc\\_cl25/examples/replaced/e01sac\\_rk.c](http://www.nag.co.uk/numeric/cl/nagdoc_cl25/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/numeric/cl/nagdoc\\_cl25/examples/replaced/e01sac\\_shep.c](http://www.nag.co.uk/numeric/cl/nagdoc_cl25/examples/replaced/e01sac_shep.c)) provides replacement call information for the Shepard's method (`nag_2d_triangular_interp` (e01sjc)).

### **nag\_2d\_scatter\_eval (e01sbc)**

Withdrawn at Mark 23.

Replaced by `nag_2d_shep_eval` (e01shc) or `nag_2d_triangular_eval` (e01skc).

See the example program `e01sac_rk.c` (see [http://www.nag.co.uk/numeric/cl/nagdoc\\_cl25/examples/replaced/e01sac\\_rk.c](http://www.nag.co.uk/numeric/cl/nagdoc_cl25/examples/replaced/e01sac_rk.c)) and `e01sac_shep.c` (see [http://www.nag.co.uk/numeric/cl/nagdoc\\_cl25/examples/replaced/e01sac\\_shep.c](http://www.nag.co.uk/numeric/cl/nagdoc_cl25/examples/replaced/e01sac_shep.c)) for full details.

### **nag\_2d\_scatter\_free (e01szc)**

Withdrawn at Mark 23.

There is no replacement for this function.

## **e04 – Minimizing or Maximizing a Function**

### **nag\_opt\_simplex (e04ccc)**

Withdrawn at Mark 24.

Replaced by `nag_opt_simplex_easy` (e04cbc).

```
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** and **max\_iter** members of the **options** structure have been introduced as the arguments **tolf** and **maxcal**, respectively. **tolx** is an additional argument to control tolerance. A new user defined function **monit** has been added to allow you to monitor the optimization process. If no monitoring is required, **monit** may be specified as **NULLFN**.

### **nag\_opt\_bounds\_no\_deriv (e04jbc)**

Scheduled for withdrawal at Mark 26.

Replaced by `nag_opt_nlp` (e04ucc).

See the example program `e04jbce.c` (see [http://www.nag.co.uk/numeric/cl/nagdoc\\_cl25/examples/replaced/e04jbce.c](http://www.nag.co.uk/numeric/cl/nagdoc_cl25/examples/replaced/e04jbce.c)) for code demonstrating how to use `nag_opt_nlp` (e04ucc) instead of `nag_opt_bounds_no_deriv` (e04jbc).

## **f01 – Matrix Operations, Including Inversion**

### **nag\_complex\_cholesky (f01bnc)**

Withdrawn at Mark 25.

Replaced by `nag_zpotrf` (f07frc).

If you were only using `nag_complex_cholesky` (f01bnc) in order to feed its results into `nag_hermitian_linear_equation_multiple_rhs` (f04awc), then the simple replacement function given further below, in the section for `nag_hermitian_linear_equation_multiple_rhs` (f04awc), will suffice. A more thorough replacement

function is given here and it will put the same values in arrays **a** and **p** as `nag_complex_cholesky` (`f01bnc`) did.

```
void f01bnc_replacement(Integer n, Complex a[], Integer tda, double p[],
                       NagError *fail)
{
  Integer i, pdb=n;
  Complex *b;

  b = NAG_ALLOC(n*n, Complex);
  /* replacement factorization routine requires the upper triangle
     to be stored for UH*U, but f01bnc expects the lower triangle
     to be stored so put the lower triangle of a into the upper
     triangle of b */
  /* nag_zge_copy */
  f16tfc(Nag_RowMajor, Nag_ConjTrans, n, n, a, tda, b, pdb, fail);
  /* factorize b */
  /* nag_zpotrf */
  f07frc(Nag_RowMajor, Nag_Upper, n, b, pdb, fail);
  /* diagonal elements to populate the p array */
  for (i = 0; i < n; ++i) p[i] = 1.0/b[i*tda+i].re;
  /* overwrite the off-diagonal upper triangle of a with U */
  /* nag_ztr_copy */
  f16tec(Nag_RowMajor, Nag_Upper, Nag_NoTrans, Nag_UnitDiag, n, b, pdb, a,
         tda, fail);
  NAG_FREE(b);
}
```

### **nag\_real\_qr (f01qcc)**

Withdrawn at Mark 25.

Replaced by `nag_dgeqrf` (`f08aec`).

The subdiagonal elements of **a** and the elements of **zeta** returned by `nag_dgeqrf` (`f08aec`) are not the same as those returned by `nag_real_qr` (`f01qcc`). Subsequent calls to `nag_real_apply_q` (`f01qdc`) or `nag_real_form_q` (`f01qec`) must also be replaced by calls to `nag_dorgqr` (`f08afc`) or `nag_dormqr` (`f08agc`) as shown below.

```
void f01qcc_replacement(Integer m, Integer n, double a[], Integer tda,
                       double zeta[], NagError *fail)
{
  /* nag_dgeqrf */
  f08aec(Nag_RowMajor, m, n, a, tda, zeta, fail);
  /* the factorization in a and zeta will be stored differently */
}
```

### **nag\_real\_apply\_q (f01qdc)**

Withdrawn at Mark 25.

Replaced by `nag_dormqr` (`f08agc`).

The following replacement is valid only if the previous call to `nag_real_qr` (`f01qcc`) has been replaced by a call to `nag_dgeqrf` (`f08aec`) as shown below. It also assumes that the second argument of

`nag_real_apply_q` (`f01qdc`) is set to `wheret = Nag_ElementsSeparate`, which is appropriate if the contents of `a` and `zeta` have not been changed after the call of `nag_real_qr` (`f01qcc`).

```
void f01qcc_replacement(Integer m, Integer n, double a[], Integer tda,
    double zeta[], NagError *fail)
{
    /* nag_dgeqrf */
    f08aec(Nag_RowMajor, m, n, a, tda, zeta, fail);
    /* the factorization in a and zeta will be stored differently */
}

void f01qdc_replacement(MatrixTranspose trans, Nag_WhereElements wheret,
    Integer m, Integer n, double a[], Integer tda, const double zeta[],
    Integer ncolb, double b[], Integer tdb, NagError *fail) {
    Nag_TransType t = (trans==NoTranspose)? Nag_NoTrans : Nag_Trans;
    /* nag_dormqr */
    f08agc(Nag_RowMajor, Nag_LeftSide, t, m, ncolb, n, a, tda, zeta,
        b, tdb, fail);
}
```

### **nag\_real\_form\_q (f01qec)**

Withdrawn at Mark 25.

Replaced by `nag_dorgqr` (`f08afc`).

The following replacement is valid only if the previous call to `nag_real_qr` (`f01qcc`) has been replaced by a call to `nag_dgeqrf` (`f08aec`) as shown below. It also assumes that the first argument of `nag_real_form_q` (`f01qec`) is set to `wheret = Nag_ElementsSeparate`, which is appropriate if the contents of `a` and `zeta` have not been changed after the call of `nag_real_qr` (`f01qcc`).

```
void f01qcc_replacement(Integer m, Integer n, double a[], Integer tda,
    double zeta[], NagError *fail)
{
    /* nag_dgeqrf */
    f08aec(Nag_RowMajor, m, n, a, tda, zeta, fail);
    /* the factorization in a and zeta will be stored differently */
}

void f01qec_replacement(Nag_WhereElements wheret, Integer m, Integer n,
    Integer ncolq, double a[], Integer tda, const double zeta[],
    NagError *fail)
{
    /* factorization performed by nag_dgeqrf (f08aec) */
    /* nag_dorgqr */
    f08afc(Nag_RowMajor, m, ncolq, n, a, tda, zeta, fail);
}
```

### **nag\_complex\_qr (f01rcc)**

Withdrawn at Mark 25.

Replaced by `nag_zgeqrf` (`f08asc`).

The subdiagonal elements of `a` and the elements of `theta` returned by `nag_zgeqrf` (`f08asc`) are not the same as those returned by `nag_complex_qr` (`f01rcc`). Subsequent calls to `nag_complex_apply_q` (`f01rdc`) or `nag_complex_form_q` (`f01rec`) must also be replaced by calls to `nag_zunmqr` (`f08auc`) or `nag_zungqr` (`f08atc`) as shown below.

```
void f01rcc_replacement(Integer m, Integer n, Complex a[], Integer tda,
    Complex theta[], NagError *fail)
{
    /* nag_zgeqrf */
    f08asc(Nag_RowMajor, m, n, a, tda, theta, fail);
    /* the factorization in a and theta will be stored differently */
}
```

### **nag\_complex\_apply\_q (f01rdc)**

Withdrawn at Mark 25.

Replaced by `nag_zunmqr` (`f08auc`).

The following replacement is valid only if the previous call to `nag_complex_qr` (`f01rcc`) has been replaced by a call to `nag_zgeqrf` (`f08asc`) as shown below. It also assumes that the second argument of `nag_complex_apply_q` (`f01rdc`) is set to `wheret = Nag_ElementsSeparate`, which is appropriate if the contents of `a` and `theta` have not been changed after the call of `nag_complex_qr` (`f01rcc`).

```
void f01rcc_replacement(Integer m, Integer n, Complex a[], Integer tda,
    Complex theta[], NagError *fail)
{
    /* nag_zgeqrf */
    f08asc(Nag_RowMajor, m, n, a, tda, theta, fail);
    /* the factorization in a and theta will be stored differently */
}

void f01rdc_replacement(MatrixTranspose trans, Nag_WhereElements wheret,
    Integer m, Integer n, Complex a[], Integer tda, const Complex theta[],
    Integer ncolb, Complex b[], Integer tdb, NagError *fail)
{
    Nag_TransType t = (trans==NoTranspose)? Nag_NoTrans : Nag_ConjTrans;
    /* nag_zunmqr */
    f08auc(Nag_RowMajor, Nag_LeftSide, t, m, ncolb, n, a, tda, theta,
        b, tdb, fail);
}
```

### **nag\_complex\_form\_q (f01rec)**

Withdrawn at Mark 25.

Replaced by `nag_zungqr` (`f08atc`).

The following replacement is valid only if the previous call to `nag_complex_qr` (`f01rcc`) has been replaced by a call to `nag_zgeqrf` (`f08asc`) as shown below. It also assumes that the first argument of `nag_complex_form_q` (`f01rec`) is set to `wheret = Nag_ElementsSeparate`, which is appropriate if the contents of `a` and `theta` have not been changed after the call of `nag_complex_qr` (`f01rcc`).

```
void f01rcc_replacement(Integer m, Integer n, Complex a[], Integer tda,
    Complex theta[], NagError *fail)
{
    /* nag_zgeqrf */
    f08asc(Nag_RowMajor, m, n, a, tda, theta, fail);
    /* the factorization in a and theta will be stored differently */
}

void f01rec_replacement(Nag_WhereElements wheret, Integer m, Integer n,
    Integer ncolq, Complex a[], Integer tda, const Complex theta[],
    NagError *fail)
{
    /* nag_zungqr */
    /* factorization performed by nag_zgeqrf (f08asc) */
    f08atc(Nag_RowMajor, m, ncolq, n, a, tda, theta, fail);
}
```

## **f02 – Eigenvalues and Eigenvectors**

### **nag\_real\_symm\_eigenvalues (f02aac)**

Scheduled for withdrawal at Mark 26.

Replaced by `nag_dsyev` (`f08fac`).

Old: `nag_real_symm_eigenvalues(n, a, tda, r, &fail);`  
 New: `nag_dsyev(Nag_RowMajor, Nag_EigVals, Nag_Lower, n, a, tda, r, &fail);`

**nag\_real\_symm\_eigensystem (f02abc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_dsyev (f08fac).

```
Old: nag_real_symm_eigensystem(n, a, tda, r, v, tdv, &fail);
New: nag_dtr_copy (Nag_RowMajor, Nag_Lower, Nag_NoTrans, Nag_NonUnitDiag, n,
                  a, tda, v, tdv, &fail);
      nag_dsyev(Nag_RowMajor, Nag_DoBoth, Nag_Lower, n, v, tdv, r, &fail);
```

If nag\_real\_symm\_eigensystem (f02abc) was called with the same array supplied for **v** and **a**, then the call to nag\_dtr\_copy (f16qec) may be omitted.

**nag\_real\_symm\_general\_eigenvalues (f02adc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_dsygv (f08sac).

```
Old: nag_real_symm_general_eigenvalues(n, a, tda, b, tdb, r, &fail);
New: nag_dsygv(Nag_RowMajor, 1, Nag_EigVals, Nag_Upper, n, a, tda, b, tdb,
              r, &fail);
```

Note that the call to nag\_dsygv (f08sac) will overwrite the upper triangles of the arrays **a** and **b** and leave the subdiagonal elements unchanged, whereas the call to nag\_real\_symm\_general\_eigenvalues (f02adc) overwrites the lower triangle and leaves the elements above the diagonal unchanged.

**nag\_real\_symm\_general\_eigensystem (f02aec)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_dsygv (f08sac).

```
Old: nag_real_symm_general_eigensystem(n, a, tda, b, tdb, r, v, tdv,
                                       &fail);
New: nag_dtr_copy (Nag_RowMajor, Nag_Upper, Nag_NoTrans, Nag_NonUnitDiag, n,
                  a, tda, v, tdv, &fail);
      nag_dsygv(Nag_RowMajor, 1, Nag_DoBoth, Nag_Upper, n, v, tdv, b, tdb,
              r, &fail);
```

Note that the call to nag\_dsygv (f08sac) will overwrite the upper triangle of the array **b** and leave the subdiagonal elements unchanged, whereas the call to nag\_real\_symm\_general\_eigensystem (f02aec) overwrites the lower triangle and leaves the elements above the diagonal unchanged. The call to nag\_dtr\_copy (f16qec) copies **a** to **v**, so **a** is left unchanged. If nag\_real\_symm\_general\_eigensystem (f02aec) was called with the same array supplied for **v** and **a**, then the call to nag\_dtr\_copy (f16qec) may be omitted.

**nag\_real\_eigenvalues (f02afc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_dgeev (f08nac).

```
Old: nag_real_eigenvalues(n, a, tda, r, iter, &fail);
New: nag_dgeev(Nag_RowMajor, Nag_NotLeftVecs, Nag_NotRightVecs, n, a, tda,
              wr, wi, vl, 1, vr, 1, &fail);
```

where **wr** and **wi** are double arrays of lengths  $n$  such that  $\mathbf{wr}[i-1] = \mathbf{r.re}[i-1]$  and  $\mathbf{wi}[i-1] = \mathbf{r.im}[i-1]$ , for  $i = 1, 2, \dots, n$ ; **vl** and **vr** are double arrays of length 1 (not used in this call); the iteration counts (returned by nag\_real\_eigenvalues (f02afc) in the array **iter**) are not available from nag\_dgeev (f08nac).

**nag\_real\_eigensystem (f02agc)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_dgeev (f08nac).

```
Old: nag_real_eigensystem(n, a, tda, r, v, tdv, iter, &fail);
New: nag_dgeev(Nag_RowMajor, Nag_NotLeftVecs, Nag_RightVecs, n, a, tda,
              wr, wi, vl, 1, vr, pdvr, &fail);
```

where **wr** and **wi** are double arrays of lengths  $n$  such that  $\mathbf{wr}[i-1] = \mathbf{r.re}[i-1]$  and  $\mathbf{wi}[i-1] = \mathbf{r.im}[i-1]$ , for  $i = 1, 2, \dots, n$ ; **vl** is a double array of length 1 (not used in this call) and **vr** is a double array of length  $\mathbf{n} \times \mathbf{n}$ ; the iteration counts (returned by `nag_real_eigensystem` (f02agc) in the array **iter**) are not available from `nag_dgeev` (f08nac).

Eigenvector information is stored differently in **vr**:

$$\mathbf{v.re}[j] = \mathbf{vr}[j] \text{ if } \mathbf{wi}[j] = 0.0.$$

$$\mathbf{v.re}[j] = \mathbf{vr}[j] \text{ and } \mathbf{v.im}[j] = \mathbf{vr}[j+1] \text{ and } \mathbf{v.re}[j+1] = \mathbf{vr}[j] \text{ and } \mathbf{v.im}[j+1] = -\mathbf{vr}[j+1] \text{ if } \mathbf{wi}[j] \neq 0 \text{ and } \mathbf{wi}[j] = -\mathbf{wi}[j+1].$$

### **nag\_hermitian\_eigenvalues (f02awc)**

Scheduled for withdrawal at Mark 26.

Replaced by `nag_zheev` (f08fnc).

Old: `nag_hermitian_eigenvalues(n, a, tda, r, &fail);`

New: `nag_zheev(Nag_RowMajor, Nag_EigVals, Nag_Lower, n, a, tda, r, &fail);`

### **nag\_hermitian\_eigensystem (f02axc)**

Scheduled for withdrawal at Mark 26.

Replaced by `nag_zheev` (f08fnc).

Old: `nag_hermitian_eigensystem(n, a, tda, r, v, tdv, &fail);`

New: `nag_ztr_copy(Nag_RowMajor, Nag_Lower, Nag_NoTrans, Nag_NonUnitDiag, n, a, tda, v, tdv, &fail);`  
`nag_zheev(Nag_RowMajor, Nag_DoBoth, Nag_Lower, n, v, tdv, r, &fail);`

If `nag_hermitian_eigensystem` (f02axc) was called with the same arrays supplied for **v** and **a**, then the call to `nag_ztr_copy` (f16tec) may be omitted.

### **nag\_real\_general\_eigensystem (f02bjc)**

Scheduled for withdrawal at Mark 26.

Replaced by `nag_dggeev` (f08wac).

Old: `nag_real_general_eigensystem(n, a, tda, b, tdb, tol, alfa, beta, wantv, v, tdv, iter, &fail);`

New: `if (wantv) jobvr = Nag_RightVecs; else jobvr = Nag_NotRightVecs;`  
`nag_dggeev(Nag_RowMajor, Nag_NotLeftVecs, jobvr, n, a, tda, b, tdb, alphas, alphas, beta, vl, tdvl, vr, tdvr, &fail);`

where **alphar** and **alphai** are double arrays of lengths  $n$  such that  $\mathbf{alphar}[i-1] = \mathbf{alfa.re}[i-1]$  and  $\mathbf{alphai}[i-1] = \mathbf{alfa.im}[i-1]$ , for  $i = 1, 2, \dots, n$ .

### **nag\_real\_svd (f02wec)**

Scheduled for withdrawal at Mark 26.

Replaced by `nag_dgesvd` (f08kbc).

Old: `nag_real_svd(m, n, a, tda, ncolb, b, tdb, wantq, q, tdq, sv, wantp, pt, tdpt, &iter, e, &failinfo, &fail);`

New: `if (wantq) jobu = Nag_AllU; else jobu = Nag_NotU;`  
`if (wantp) jobvt = Nag_AllVT; else jobvt = Nag_NotVT;`  
`nag_dgesvd(Nag_RowMajor, jobu, jobvt, m, n, a, tda, sv, q, tdq, pt, tdpt, work, &fail);`

**work** must be a one-dimensional double array of length  $\min(\mathbf{m}, \mathbf{n})$ ; the iteration count (returned by `nag_real_svd` (f02wec) in the argument **iter**) is not available from `nag_dgesvd` (f08kbc).

Please note that the facility to return  $Q^T B$  is not provided so arguments **ncolb** and **b** are not required. Instead, `nag_dgesvd` (f08kbc) has an option to return the entire  $\mathbf{m} * \mathbf{m}$  orthogonal matrix  $Q$ , referred to as **u** in its documentation, through its 8th argument.

**nag\_complex\_svd (f02xec)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_zgesvd (f08kpc).

```
Old: nag_complex_svd(m, n, a, tda, ncolb, b, tdb, wantq, q, tdq, sv, wantp,
                    ph, tdph, &iter, e, &failinfo, &fail);
New: if (wantq) jobu = Nag_AllU; else jobu = Nag_NotU;
     if (wantp) jobvt = Nag_AllVT; else jobvt = Nag_NotVT;
     nag_zgesvd(Nag_RowMajor, jobu, jobvt, m, n, a, tda, sv, q, tdq,
               ph, tdph, rwork, &fail);
```

**rwork** must be a one-dimensional double array of length  $\min(\mathbf{m}, \mathbf{n})$ ; the iteration count (returned by nag\_complex\_svd (f02xec) in the argument **iter**) is not available from nag\_zgesvd (f08kpc).

Please note that the facility to return  $Q^H B$  is not provided so arguments **ncolb** and **b** are not required. Instead, nag\_zgesvd (f08kpc) has an option to return the entire  $\mathbf{m} * \mathbf{m}$  unitary matrix  $Q$ , referred to as **u** in its documentation, through its 8th argument.

**f03 – Determinants****nag\_real\_cholesky (f03aec)**

Withdrawn at Mark 25.

Replaced by nag\_dpotrf (f07fdc) and nag\_det\_real\_sym (f03bfc).

```
void f03aec_replacement(Integer n, double a[], Integer tda,
                        double p[], double *detf, Integer *dete, NagError *fail)
{
    /* nag_dpotrf */
    f07fdc(Nag_RowMajor, Nag_Upper, n, a, tda, fail);
    /* nag_det_real_sym */
    f03bfc(Nag_RowMajor, n, a, tda, detf, dete, fail);
    /* p is not written to */
    /* factorization in a will be different */
}
```

nag\_dpotrf (f07fdc) performs the Cholesky factorization and nag\_det\_real\_sym (f03bfc) calculates the determinant from the factored form.

**Note:** subsequent solution of linear systems using the Cholesky factorization performed by nag\_dpotrf (f07fdc) should be performed using nag\_dpots (f07fec).

**nag\_real\_lu (f03afc)**

Withdrawn at Mark 25.

Replaced by nag\_dgetrf (f07adc) and nag\_det\_real\_gen (f03bac).

```
void f03afc_replacement(Integer n, double a[], Integer tda,
                        Integer pivot[], double *detf, Integer *dete, NagError *fail)
{
    /* nag_dgetrf */
    f07adc(Nag_RowMajor, n, n, a, tda, pivot, fail);
    /* nag_det_real_gen */
    f03bac(Nag_RowMajor, n, a, tda, pivot, detf, dete, fail);
    /* the factorization in a will be different */
    /* the array pivot will be different */
}
```

**Note:** subsequent solution of linear systems using the  $LU$  factorization performed by nag\_dgetrf (f07adc) should be performed using nag\_dgetrs (f07aec).



**nag\_complex\_lu (f03ahc)**

Withdrawn at Mark 25.

Replaced by nag\_zgetrf (f07arc) and nag\_det\_complex\_gen (f03bnc).

```
void f03ahc_replacement(Integer n, Complex a[], Integer tda,
    Integer pivot[], Complex *det, Integer *dete, NagError *fail)
{
    Complex d={0,0};
    Integer id[2]={0,0};
    /* nag_zgetrf */
    f07arc(Nag_RowMajor, n, n, a, tda, pivot, fail);
    /* nag_det_complex_gen */
    f03bnc(Nag_RowMajor, n, a, tda, pivot, &d, id, fail);
    /* Bring real and imaginary parts to a common scale */
    *dete = MAX(id[0],id[1]);
    det->re = ldexp(d.re,id[0]-*dete);
    det->im = ldexp(d.im,id[1]-*dete);
    /* the factorization in a will be different */
}

```

nag\_zgetrf (f07arc) performs the *LU* factorization and nag\_det\_complex\_gen (f03bnc) calculates the determinant from the factored form.

**Note:** the details of the *LU* factorization performed by nag\_zgetrf (f07arc) differ from those performed by nag\_complex\_lu (f03ahc); subsequent solution of linear systems using the *LU* factorization performed by nag\_zgetrf (f07arc) should be performed using nag\_zgets (f07asc). The determinant returned by nag\_det\_complex\_gen (f03bnc) independently scales the real and imaginary parts whereas the determinant returned by nag\_complex\_lu (f03ahc) used a single scaling factor.

**f04 – Simultaneous Linear Equations**

The factorization and solution of a positive definite linear system can be handled by calls to functions from Chapter f07.

**nag\_complex\_lin\_eqn\_mult\_rhs (f04adc)**

Withdrawn at Mark 25.

Replaced by nag\_complex\_gen\_lin\_solve (f04cac).

```
void f04adc_replacement(Integer n, Integer nrhs,
    Complex a[], Integer tda, const Complex b[], Integer tdb,
    Complex x[], Integer tdx, NagError *fail)
{
    Integer *ipiv;
    double rcond, errbnd;

    ipiv = NAG_ALLOC(n, Integer);
    /* nag_zge_copy */
    f16tfc(Nag_RowMajor, Nag_NoTrans, n, nrhs, b, tdb, x, tdx, fail);
    /* nag_complex_gen_lin_solve */
    f04cac(Nag_RowMajor, n, nrhs, a, tda, ipiv, x,
    tdx, &rcond, &errbnd, fail);
    /* The factorization in a will be different */
    /* Error codes will be different */
    /* Condition number and error bounds are available to you */
    NAG_FREE(ipiv);
}

```

**nag\_real\_cholesky\_solve\_mult\_rhs (f04agc)**

Withdrawn at Mark 25.

Replaced by nag\_dpotsr (f07fec).

It is assumed that the matrix has been factorized by a call to `nag_dpotrf` (f07fdc) rather than `nag_real_cholesky` (f03aec). The array `p` is no longer required.

```
void f03aec_replacement(Integer n, double a[], Integer tda,
    double p[], double *detf, Integer *dete, NagError *fail)
{
    /* nag_dpotrf */
    f07fdc(Nag_RowMajor, Nag_Upper, n, a, tda, fail);
    /* nag_det_real_sym */
    f03bfc(Nag_RowMajor, n, a, tda, detf, dete, fail);
    /* p is not used */
    /* the factorization in a will be different */
}

void f04agc_replacement(Integer n, Integer nrhs, double a[],
    Integer tda, double p[], const double b[], Integer tdb, double x[],
    Integer tdx, NagError *fail)
{
    /* nag_dge_copy */
    f16qfc(Nag_RowMajor, Nag_NoTrans, n, nrhs, b, tdb, x, tdx, fail);
    /* nag_dpotrs */
    f07fec(Nag_RowMajor, Nag_Upper, n, nrhs, a, tda, x, tdx, fail);
    /* p is not used */
}

```

### **nag\_real\_lu\_solve\_mult\_rhs (f04ajc)**

Withdrawn at Mark 25.

Replaced by `nag_dgetrs` (f07aec).

It is assumed that the matrix has been factorized by a call to `nag_dgetrf` (f07adc) rather than `nag_real_lu` (f03afc).

```
void f03afc_replacement(Integer n, double a[], Integer tda,
    Integer pivot[], double *detf, Integer *dete, NagError *fail)
{
    /* nag_dgetrf */
    f07adc(Nag_RowMajor, n, n, a, tda, pivot, fail);
    /* nag_det_real_gen */
    f03bac(Nag_RowMajor, n, a, tda, pivot, detf, dete, fail);
    /* the call to f03bac is not needed if you don't want determinants */
}

void f04ajc_replacement(Integer n, Integer nrhs, const double a[],
    Integer tda, const Integer pivot[], double b[], Integer tdb,
    NagError *fail)
{
    /* nag_dgetrs */
    f07aec(Nag_RowMajor, Nag_NoTrans, n, nrhs, a, tda, pivot, b, tdb, fail);
}

```

**nag\_complex\_lu\_solve\_mult\_rhs (f04akc)**

Withdrawn at Mark 25.

Replaced by nag\_zgetrs (f07asc).

```
void f03ahc_replacement(Integer n, Complex a[], Integer tda,
    Integer pivot[], Complex *det, Integer *dete, NagError *fail)
{
    Complex d={0,0};
    Integer id[2]={0,0};
    /* nag_zgetrf */
    f07arc(Nag_RowMajor, n, n, a, tda, pivot, fail);
    /* nag_det_complex_gen */
    f03bnc(Nag_RowMajor, n, a, tda, pivot, &d, id, fail);
    /* Bring real and imaginary parts to a common scale */
    *dete = MAX(id[0],id[1]);
    det->re = ldexp(d.re,id[0]-*dete);
    det->im = ldexp(d.im,id[1]-*dete);
    /* the factorization in a will be different */
}

void f04akc_replacement(Integer n, Integer nrhs, const Complex a[],
    Integer tda, const Integer pivot[], Complex b[], Integer tdb,
    NagError *fail)
{
    /* nag_zgetrs */
    f07asc(Nag_RowMajor, Nag_NoTrans, n, nrhs, a, tda, pivot, b, tdb, fail );
}

```

It is assumed that the matrix has been factorized by a call to nag\_zgetrf (f07arc) rather than nag\_complex\_lu (f03ahc).

**nag\_real\_lin\_eqn (f04arc)**

Withdrawn at Mark 25.

Replaced by nag\_real\_gen\_lin\_solve (f04bac).

```
void f04arc_replacement(Integer n, double a[], Integer tda,
    const double b[], double x[], NagError *fail)
{
    Integer *ipiv;
    double rcond, errbnd;

    ipiv = NAG_ALLOC(n, Integer);
    /* nag_dge_copy */
    f16qfc(Nag_RowMajor, Nag_NoTrans, n, 1, b, 1, x, 1, fail);
    /* nag_real_gen_lin_solve */
    f04bac(Nag_RowMajor, n, 1, a, tda, ipiv, x, 1,
    &rcond, &errbnd, fail);
    /* The factorization in a will be different */
    /* Error codes will be different */
    /* Condition number and error bounds are available to you */
    NAG_FREE(ipiv);
}

```

**nag\_hermitian\_lin\_eqn\_mult\_rhs (f04awc)**

Withdrawn at Mark 25.

Replaced by nag\_zpotrs (f07fsc).

```
void f01bnc_replacement(Integer n, Complex a[], Integer tda,
    double p[], NagError *fail)
{
    /* nag_zpotrf */
    f07frc(Nag_RowMajor, Nag_Lower, n, a, tda, fail);
}

void f04awc_replacement(Integer n, Integer nrhs, const Complex a[],
    Integer tda, const double p[], const Complex b[],
    Integer tdb, Complex x[], Integer tdx, NagError *fail)
{
    /* nag_zge_copy */
    f16tfc(Nag_RowMajor, Nag_NoTrans, n, nrhs, b, tdb, x, tdx, fail);
    /* nag_zpotrs */
    f07fsc(Nag_RowMajor, Nag_Lower, n, nrhs, a, tda, x, tdx, fail);
}
```

Note that the preceding call to nag\_complex\_cholesky (f01bnc) has been replaced by nag\_zpotrf (f07frc).

**f06 – Linear Algebra Support Functions**

The functions in Chapter f16 provide greater functionality than their corresponding functions in Chapter f06. The essential differences are:

The **order** argument. This provides the flexibility to operate on matrix data stored in row or column major order.

The addition of the **fail** argument to trap data errors. The f06 functions used to abort noisily.

The enumeration types and members use NAG\_ as the prefix. This is to guard against accidental use of non-NAG enums.

Scale factors have been introduced in some functions. For example nag\_dtrmv (f16pfc) has an extra argument, **alpha** which was not present in the corresponding old\_dtrmv (f06pfc) function.

**old\_dgemv (f06pac)**

Withdrawn at Mark 23.

Replaced by nag\_dgemv (f16pac).

**old\_dgbmv (f06pbc)**

Withdrawn at Mark 23.

Replaced by nag\_dgbmv (f16pbc).

**old\_dsymv (f06pcc)**

Withdrawn at Mark 23.

Replaced by nag\_dsymv (f16pcc).

**old\_dsblmv (f06pdc)**

Withdrawn at Mark 23.

Replaced by nag\_dsblmv (f16pdc).

**old\_dspmv (f06pec)**

Withdrawn at Mark 23.

Replaced by nag\_dspmv (f16pec).

**old\_dtrmv (f06pfc)**

Withdrawn at Mark 23.  
Replaced by nag\_dtrmv (f16pfc).

**old\_dtbmv (f06pgc)**

Withdrawn at Mark 23.  
Replaced by nag\_dtbmv (f16pgc).

**old\_dtpmv (f06phc)**

Withdrawn at Mark 23.  
Replaced by nag\_dtpmv (f16phc).

**old\_dtrsv (f06pjc)**

Withdrawn at Mark 23.  
Replaced by nag\_dtrsv (f16pjc).

**old\_dtbsv (f06pkc)**

Withdrawn at Mark 23.  
Replaced by nag\_dtbsv (f16pkc).

**old\_dtpsv (f06plc)**

Withdrawn at Mark 23.  
Replaced by nag\_dtpsv (f16plc).

**old\_dger (f06pmc)**

Withdrawn at Mark 23.  
Replaced by nag\_dger (f16pmc).

**old\_dsyrr (f06ppc)**

Withdrawn at Mark 23.  
Replaced by nag\_dsyrr (f16ppc).

**old\_dspr (f06pqc)**

Withdrawn at Mark 23.  
Replaced by nag\_dspr (f16pqc).

**old\_dsyrr2 (f06prc)**

Withdrawn at Mark 23.  
Replaced by nag\_dsyrr2 (f16prc).

**old\_dspr2 (f06psc)**

Withdrawn at Mark 23.  
Replaced by nag\_dspr2 (f16psc).

**old\_zgemv (f06sac)**

Withdrawn at Mark 23.  
Replaced by nag\_zgemv (f16sac).

**old\_zgbmv (f06sbc)**

Withdrawn at Mark 23.  
Replaced by nag\_zgbmv (f16sbc).

**old\_zhemv (f06scc)**

Withdrawn at Mark 23.  
Replaced by nag\_zhemv (f16scc).

**old\_zhbmV (f06sdc)**

Withdrawn at Mark 23.  
Replaced by nag\_zhbmV (f16sdc).

**old\_zhpmv (f06sec)**

Withdrawn at Mark 23.  
Replaced by nag\_zhpmv (f16sec).

**old\_ztrmv (f06sfc)**

Withdrawn at Mark 23.  
Replaced by nag\_ztrmv (f16sfc).

**old\_ztbmv (f06sgc)**

Withdrawn at Mark 23.  
Replaced by nag\_ztbmv (f16sgc).

**old\_ztpmv (f06shc)**

Withdrawn at Mark 23.  
Replaced by nag\_ztpmv (f16shc).

**old\_ztrsv (f06sjc)**

Withdrawn at Mark 23.  
Replaced by nag\_ztrsv (f16sjc).

**old\_ztbsv (f06skc)**

Withdrawn at Mark 23.  
Replaced by nag\_ztbsv (f16skc).

**old\_ztpsv (f06slc)**

Withdrawn at Mark 23.  
Replaced by nag\_ztpsv (f16slc).

**old\_zgeru (f06smc)**

Withdrawn at Mark 23.  
Replaced by nag\_zger (f16smc).

**old\_zgerc (f06snc)**

Withdrawn at Mark 23.  
Replaced by nag\_zger (f16smc).

**old\_zher (f06spc)**

Withdrawn at Mark 23.  
Replaced by nag\_zher (f16spc).

**old\_zhpr (f06sqc)**

Withdrawn at Mark 23.  
Replaced by nag\_zhpr (f16sqc).

**old\_zher2 (f06src)**

Withdrawn at Mark 23.  
Replaced by nag\_zher2 (f16src).

**old\_zhpr2 (f06ssc)**

Withdrawn at Mark 23.  
Replaced by nag\_zhpr2 (f16ssc).

**old\_dgemm (f06yac)**

Withdrawn at Mark 23.  
Replaced by nag\_dgemm (f16yac).

**old\_dsymm (f06ycc)**

Withdrawn at Mark 23.  
Replaced by nag\_dsymm (f16ycc).

**old\_dtrmm (f06yfc)**

Withdrawn at Mark 23.  
Replaced by nag\_dtrmm (f16yfc).

**old\_dtrsm (f06yjc)**

Withdrawn at Mark 23.  
Replaced by nag\_dtrsm (f16yjc).

**old\_dsyrk (f06ypc)**

Withdrawn at Mark 23.  
Replaced by nag\_dsyrk (f16ypc).

**old\_dsy2k (f06yrc)**

Withdrawn at Mark 23.  
Replaced by nag\_dsy2k (f16yrc).

**old\_zgemm (f06zac)**

Withdrawn at Mark 23.  
Replaced by nag\_zgemm (f16zac).

**old\_zhemm (f06zcc)**

Withdrawn at Mark 23.  
Replaced by nag\_zhemm (f16zcc).

**old\_ztrmm (f06zfc)**

Withdrawn at Mark 23.  
Replaced by nag\_ztrmm (f16zfc).

**old\_ztrsm (f06zjc)**

Withdrawn at Mark 23.  
Replaced by nag\_ztrsm (f16zjc).

**old\_zherk (f06zpc)**

Withdrawn at Mark 23.  
Replaced by nag\_zherk (f16zpc).

**old\_zher2k (f06zrc)**

Withdrawn at Mark 23.  
Replaced by nag\_zher2k (f16zrc).

**old\_zsymm (f06ztc)**

Withdrawn at Mark 23.  
Replaced by nag\_zsymm (f16ztc).

**old\_zsyrk (f06zuc)**

Withdrawn at Mark 23.  
Replaced by nag\_zsyrk (f16zuc).

**old\_zsyr2k (f06zwc)**

Withdrawn at Mark 23.  
Replaced by nag\_zsyr2k (f16zwc).

**g01 – Simple Calculations on Statistical Data****nag\_summary\_stats\_1var (g01aac)**

Scheduled for withdrawal at Mark 26.  
Replaced by nag\_summary\_stats\_onevar (g01atc).

```
Old:
    /* nag_summary_stats_1var (g01aac) */
    nag_summary_stats_1var(n, x, wt, &nvalid, &xmean, &xsd, &xskew,
                          &xkurt,&xmin, &xmax, &wsum, &fail);

New:
    /* nag_summary_stats_onevar (g01atc) */
    pn = 0;
    nag_summary_stats_onevar(n, x, wt, &pn, &xmean, &xsd, &xskew, &xkurt,
                          &xmin, &xmax, rcomm, &fail);

    nvalid = pn;
    wtsum = rcomm[0];
```

**nag\_deviates\_normal\_dist (g01cec)**

Withdrawn at Mark 24.  
Replaced by nag\_deviates\_normal (g01fac).

```
Old: x = nag_deviates_normal_dist(p, &fail);
New: x = nag_deviates_normal(Nag_LowerTail, p, &fail);
```



## g02 – Correlation and Regression Analysis

### nag\_full\_step\_regsn\_monit (g02ewc)

Withdrawn at Mark 25.

Replaced by nag\_full\_step\_regsn\_monfun (g02efh) (see **monfun** in nag\_full\_step\_regsn (g02efc)).

Old: nag\_full\_step\_regsn\_monit(flag, var, val, &fail)

New: nag\_full\_step\_regsn\_monfun(flag, var, val, &fail)

Note: it is unlikely that you will need to call this function directly. Rather it will be supplied as a function argument to nag\_full\_step\_regsn (g02efc) when monitoring information is required.

## g05 – Random Number Generators

### nag\_random\_continuous\_uniform (g05cac)

Withdrawn at Mark 24.

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)

Withdrawn at Mark 24.

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 from a sequence produced by a generator initialized by nag\_random\_init\_repeatabe (g05cbc), even if the same value for **i** is used.

Note: it may still be necessary to call nag\_random\_init\_repeatabe (g05cbc) rather than the replacement function nag\_rand\_init\_repeatabe (g05kfc) when using nag\_multid\_quad\_monte\_carlo\_1 (d01xbc). See Section 10 in nag\_multid\_quad\_monte\_carlo\_1 (d01xbc) for additional information.

**nag\_random\_init\_nonrepeatable (g05ccc)**

Withdrawn at Mark 24.

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.

Note: it may still be necessary to call `nag_random_init_nonrepeatable (g05ccc)` rather than the replacement function `nag_rand_init_nonrepeatable (g05kgc)` when using `nag_multid_quad_monte_carlo_1 (d01xbc)`. See Section 10 in `nag_multid_quad_monte_carlo_1 (d01xbc)` for additional information.

**nag\_save\_random\_state (g05cfc)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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  $\mathbf{m} \times (\mathbf{m} + 1) + 1$ . In contrast to the equivalent argument in `nag_ref_vec_multi_normal (g05eac)`, this array must be allocated in the calling program.

### **nag\_ref\_vec\_poisson (g05ecc)**

Withdrawn at Mark 24.

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)`, must be allocated in the calling program in contrast to 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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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) must be allocated in the calling program in contrast to 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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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_GenerateFromReference;
    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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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_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_garchGJR (g05pfc)` is likely to be different from those produced by `nag_generate_garchGJR (g05hmc)`.

### **nag\_rngs\_basic (g05kac)**

Withdrawn at Mark 24.

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);
```

The old function `nag_rngs_basic` (g05kac) returns a single variate at a time, whereas the new function `nag_rand_basic` (g05sac) returns a vector of **n** values in one go.

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.

### **nag\_rngs\_init\_repeatabe (g05kbc)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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

### **nag\_rngs\_logical (g05kec)**

Withdrawn at Mark 24.

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);
```

The old function `nag_rngs_logical` (g05kec) returns a single variate at a time, whereas the new function `nag_rand_logical` (g05tbc) returns a vector of **n** values in one go.

The Integer array **state** in the call to `nag_rand_logical` (g05tbc) contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_logical` (g05tbc) 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.

**nag\_rngs\_normal (g05lac)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_students\_t (g05lbc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to `nag_rand_students_t (g05snc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_students_t (g05snc)` 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.

**nag\_rngs\_chi\_sq (g05lcc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to `nag_rand_chi_sq (g05sdc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_chi_sq (g05sdc)` 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.

**nag\_rngs\_f (g05ldc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to `nag_rand_f (g05shc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_f (g05shc)` 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.

**nag\_rngs\_beta (g05lec)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_gamma (g05lfc)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_uniform (g05lgc)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_triangular (g05lhc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_triangular (g05spc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_triangular (g05spc) 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.

**nag\_rngs\_exp (g05ljc)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_lognormal (g05lkc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_lognormal (g05smc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_lognormal (g05smc) 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.

**nag\_rngs\_cauchy (g05llc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_cauchy (g05scc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_cauchy (g05scc) 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.

**nag\_rngs\_weibull (g05lmc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_weibull (g05ssc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_weibull (g05ssc) 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.

**nag\_rngs\_logistic (g05lnc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_logistic (g05slc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_logistic (g05slc) 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.

**nag\_rngs\_von\_mises (g05lpc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_von\_mises (g05src) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_von\_mises (g05src) 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.

**nag\_rngs\_exp\_mix (g05lqc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_exp\_mix (g05sgc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_exp\_mix (g05sgc) 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.

**nag\_rngs\_matrix\_multi\_students\_t (g05lxc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_matrix\_multi\_students\_t (g05ryc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_matrix\_multi\_students\_t (g05ryc) 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.

**nag\_rgsn\_matrix\_multi\_normal (g05lyc)**

Withdrawn at Mark 24.

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);
```

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.



**nag\_rngs\_multi\_normal (g05lzc)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_discrete\_uniform (g05mac)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_geom (g05mbc)**

Withdrawn at Mark 24.

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\_geom (g05mbc) returned the number of trials required to get the first success, whereas nag\_rand\_geom (g05tcc) returns the number of failures before the first success, therefore the value returned by nag\_rand\_geom (g05tcc) is one less than the equivalent value returned from nag\_rngs\_geom (g05mbc).

The Integer array **state** in the call to nag\_rand\_geom (g05tcc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_geom (g05tcc) 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.

**nag\_rngs\_neg\_bin (g05mcc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_neg\_bin (g05thc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_neg\_bin (g05thc) 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.

**nag\_rngs\_logarithmic (g05mdc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_logarithmic (g05tfc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_logarithmic (g05tfc) 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.

**nag\_rngs\_compd\_poisson (g05mec)**

Withdrawn at Mark 24.

Replaced by nag\_rand\_compd\_poisson (g05tkc).

Old:

```
/* nag_rngs_compd_poisson (g05mec) */
nag_rngs_compd_poisson(m,vlamda,x,igen,iseed,&fail);
```

New:

```
/* nag_rand_compd_poisson (g05tkc) */
nag_rand_compd_poisson(m,vlamda,state,x,&fail);
```

The Integer array **state** in the call to nag\_rand\_compd\_poisson (g05tkc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_compd\_poisson (g05tkc) 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.

**nag\_rngs\_binomial (g05mjc)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_poisson (g05mkc)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_hypergeometric (g05mlc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_hypergeometric (g05tec) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_hypergeometric (g05tec) 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.

**nag\_rngs\_gen\_multinomial (g05mrc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_gen\_multinomial (g05tgc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_gen\_multinomial (g05tgc) 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.

**nag\_rngs\_gen\_discrete (g05mzc)**

Withdrawn at Mark 24.

Replaced by nag\_rand\_gen\_discrete (g05tdc).

Old:

```
/* nag_rngs_gen_discrete (g05mzc) */
nag_rngs_gen_discrete(mode,p,np,ipl,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,ipl,itype,r,lr,state,x,&fail);
```

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.

**nag\_rngs\_permute (g05nac)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_sample (g05nbc)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_arma\_time\_series (g05pac)**

Withdrawn at Mark 24.

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);
```

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.

**nag\_rngs\_varma\_time\_series (g05pcc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to `nag_rand_varma (g05pjc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_varma (g05pjc)` 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.

### **nag\_rngs\_orthog\_matrix (g05qac)**

Withdrawn at Mark 24.

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);
}
```

The Integer array **state** in the call to `nag_rand_orthog_matrix (g05pxc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_orthog_matrix (g05pxc)` 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.

### **nag\_rngs\_corr\_matrix (g05qbc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to `nag_rand_corr_matrix (g05pyc)` contains information on the base generator being used. This array must have been initialized prior to calling `nag_rand_corr_matrix (g05pyc)` 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.



**nag\_rngs\_2\_way\_table (g05qdc)**

Withdrawn at Mark 24.

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);
}
```

The Integer array **state** in the call to nag\_rand\_2\_way\_table (g05pzc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_2\_way\_table (g05pzc) 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.

**nag\_rngs\_copula\_normal (g05rac)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_copula\_normal (g05rdc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_copula\_normal (g05rdc) 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.

**nag\_rngs\_copula\_students\_t (g05rbc)**

Withdrawn at Mark 24.

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);
```

The Integer array **state** in the call to nag\_rand\_copula\_students\_t (g05rcc) contains information on the base generator being used. This array must have been initialized prior to calling nag\_rand\_copula\_students\_t (g05rcc) 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.

**nag\_quasi\_random\_uniform (g05yac)**

Withdrawn at Mark 24.

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)**

Withdrawn at Mark 24.

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.

**g10 – Smoothing in Statistics****nag\_kernel\_density\_estim (g10bac)**

Scheduled for withdrawal at Mark 26.

Replaced by nag\_kernel\_density\_gauss (g10bbc).

Old: nag\_kernel\_density\_estim(n, x, window, low, high, ns, smooth, t, &fail);

New: assert(rcomm = NAG\_ALLOC(ns+20,double));  
 nag\_kernel\_density\_gauss(n, x, Nag\_WindowSupplied, &window, &low, &high, ns,  
 smooth, t, Nag\_TRUE, rcomm, &fail);

**x02 – Machine Constants****nag\_underflow\_flag (X02DAC)**

Withdrawn at Mark 24.

There is no replacement for this function.

**nag\_real\_arithmetic\_rounds (X02DJC)**

Withdrawn at Mark 24.

There is no replacement for this function.

**x04 – Input/Output Utilities**

**nag\_example\_file\_io (x04aec)**

Withdrawn at Mark 25.

There is no replacement for this function.

---