NAG Toolbox

nag_opt_uncon_conjgrd_comp (e04dg)

1 Purpose

nag_opt_uncon_conjgrd_comp (e04dg) minimizes an unconstrained nonlinear function of several variables using a pre-conditioned, limited memory quasi-Newton conjugate gradient method. First derivatives (or an 'acceptable' finite difference approximation to them) are required. It is intended for use on large scale problems.

2 Syntax

```
[iter, objf, objgrd, x, user, lwsav, iwsav, rwsav, ifail] =
nag_opt_uncon_conjgrd_comp(objfun, x, lwsav, iwsav, rwsav, 'n', n, 'user', user)
[iter, objf, objgrd, x, user, lwsav, iwsav, rwsav, ifail] = e04dg(objfun, x,
lwsav, iwsav, rwsav, 'n', n, 'user', user)
```

Before calling nag_opt_uncon_conjgrd_comp (e04dg), or the option setting function nag_opt_uncon_conjgrd_option_string (e04dk), function nag_opt_init (e04wb) **must** be called.

3 Description

nag_opt_uncon_conjgrd_comp (e04dg) is designed to solve unconstrained minimization problems of the form

$$\underset{x \in R^n}{\text{minimize}} F(x) \quad \text{subject to} \quad -\infty \leq x \leq \infty,$$

where x is an n-element vector.

You must supply an initial estimate of the solution.

For maximum reliability, it is preferable to provide all first partial derivatives. If all of the derivatives cannot be provided, you are recommended to obtain approximate values (using finite differences) by calling nag_opt_estimate_deriv (e04xa) from within **objfun**.

The method used by nag_opt_uncon_conjgrd_comp (e04dg) is described in Section 11.

4 References

Gill P E and Murray W (1979) Conjugate-gradient methods for large-scale nonlinear optimization *Technical Report SOL 79-15* Department of Operations Research, Stanford University

Gill P E, Murray W and Wright M H (1981) *Practical Optimization* Academic Press

5 Parameters

5.1 Compulsory Input Parameters

1: **objfun** – SUBROUTINE, supplied by the user.

objfun must calculate the objective function $F(x)$ and possibly its gradient as well for a specified n-element vector x.

```
[mode, objf, objgrd, user] = objfun(mode, n, x, nstate, user)
```

Input Parameters

1: **mode** – INTEGER

Indicates which values must be assigned during each call of **objfun**. Only the following values need be assigned:

mode $= 0$
objf.

mode $= 2$
objf and **objgrd**.

2: **n** – INTEGER

n, the number of variables.

3: **x**(**n**) – REAL (KIND=nag_wp) array

x, the vector of variables at which the objective function and its gradient are to be evaluated.

4: **nstate** – INTEGER

Will be 1 on the first call of **objfun** by nag_opt_uncon_conjgrd_comp (e04dg), and 0 for all subsequent calls. Thus, you may wish to test, **nstate** within **objfun** in order to perform certain calculations once only. For example, you may read data or initialize global variables when **nstate** $= 1$.

5: **user** – INTEGER array

objfun is called from nag_opt_uncon_conjgrd_comp (e04dg) with the object supplied to nag_opt_uncon_conjgrd_comp (e04dg).

Output Parameters

1: **mode** – INTEGER

May be set to a negative value if you wish to terminate the solution to the current problem, and in this case nag_opt_uncon_conjgrd_comp (e04dg) will terminate with **ifail** set to **mode**.

2: **objf** – REAL (KIND=nag_wp)

The value of the objective function at x.

3: **objgrd**(**n**) – REAL (KIND=nag_wp) array

If **mode** $= 2$, **objgrd**(i) must contain the value of $\dfrac{\partial F}{\partial x_i}$ evaluated at x, for $i = 1, 2, \ldots, n$.

4: **user** – INTEGER array

Note: **objfun** should be tested separately before being used in conjunction with nag_opt_uncon_conjgrd_comp (e04dg). See also the description of the optional parameter **Verify**.

2: **x**(**n**) – REAL (KIND=nag_wp) array

An initial estimate of the solution.

3: **lwsav**(**120**) – LOGICAL array
4: **iwsav**(**610**) – INTEGER array
5: **rwsav**(**475**) – REAL (KIND=nag_wp) array

The arrays **lwsav**, **iwsav** and **rwsav** **must not** be altered between calls to any of the functions nag_opt_uncon_conjgrd_comp (e04dg), nag_opt_uncon_conjgrd_option_string (e04dk) or nag_opt_init (e04wb).

5.2 Optional Input Parameters

1: **n** – INTEGER

Default: the dimension of the array **x**.

n, the number of variables.

Constraint: $\mathbf{n} > 0$.

2: **user** – INTEGER array

user is not used by nag_opt_uncon_conjgrd_comp (e04dg), but is passed to **objfun**. Note that for large objects it may be more efficient to use a global variable which is accessible from the m-files than to use **user**.

5.3 Output Parameters

1: **iter** – INTEGER

The total number of iterations performed.

2: **objf** – REAL (KIND=nag_wp)

The value of the objective function at the final iterate.

3: **objgrd**(**n**) – REAL (KIND=nag_wp) array

The gradient of the objective function at the final iterate (or its finite difference approximation).

4: **x**(**n**) – REAL (KIND=nag_wp) array

The final estimate of the solution.

5: **user** – INTEGER array

6: **lwsav**(**120**) – LOGICAL array
7: **iwsav**(**610**) – INTEGER array
8: **rwsav**(**475**) – REAL (KIND=nag_wp) array

9: **ifail** – INTEGER

ifail $= 0$ unless the function detects an error (see Section 5).

nag_opt_uncon_conjgrd_comp (e04dg) returns with **ifail** $= 0$ if the following three conditions are satisfied:

(i) $F_{k-1} - F_k < \tau_F(1 + |F_k|)$

(ii) $\|x_{k-1} - x_k\| < \sqrt{\tau_F}(1 + \|x_k\|)$

(iii) $\|g_k\| \leq \sqrt[3]{\tau_F}(1 + |F_k|)$ or $\|g_k\| < \epsilon_A$

where τ_F is the value of the optional parameter **Optimality Tolerance** (default value $= \epsilon^{0.8}$) and ϵ_A is the absolute error associated with computing the objective function.

For a full discussion on termination criteria see Chapter 8 of Gill *et al.* (1981).

6 Error Indicators and Warnings

Note: nag_opt_uncon_conjgrd_comp (e04dg) may return useful information for one or more of the following detected errors or warnings.

Errors or warnings detected by the function:

ifail < 0 (*warning*)

A negative value of **ifail** indicates an exit from nag_opt_uncon_conjgrd_comp (e04dg) because you set **mode** < 0 in **objfun**. The value of **ifail** will be the same as your setting of **mode**.

ifail $= 1$

Not used by this function.

ifail $= 2$

Not used by this function.

ifail $= 3$

The limiting number of iterations (as determined by the optional parameter **Iteration Limit** (default value $= \max(50, 5n)$) has been reached.

If the algorithm appears to be making satisfactory progress, then optional parameter **Iteration Limit** may be too small. If so, increase its value and rerun nag_opt_uncon_conjgrd_comp (e04dg). If the algorithm seems to be making little or no progress, then you should check for incorrect gradients as described under **ifail** $= 7$.

ifail $= 4$

The computed upper bound on the step length taken during the linesearch was too small. A rerun with an increased value of the optional parameter **Maximum Step Length** (ρ say) may be successful unless $\rho \geq 10^{20}$ (the default value), in which case the current point cannot be improved upon.

ifail $= 5$

Not used by this function.

ifail $= 6$ (*warning*)

The conditions for an acceptable solution (see argument **ifail** in Section 5) have not all been met, but a lower point could not be found.

If **objfun** computes the objective function and its gradient correctly, then this may occur because an overly stringent accuracy has been requested, i.e., the value of the optional parameter **Optimality Tolerance** (default value $= \epsilon^{0.8}$) is too small or if $\alpha_k \simeq 0$. In this case you should apply the three tests described under **ifail** $= 0$ to determine whether or not the final solution is acceptable. For a discussion of attainable accuracy see Gill *et al.* (1981).

If many iterations have occurred in which essentially no progress has been made or nag_opt_uncon_conjgrd_comp (e04dg) has failed to move from the initial point, **objfun** may be incorrect. You should refer to the comments below under **ifail** $= 7$ and check the gradients using the optional parameter **Verify** (default value $= 0$). Unfortunately, there may be small errors in the objective gradients that cannot be detected by the verification process. Finite difference approximations to first derivatives are catastrophically affected by even small inaccuracies.

ifail $= 7$ (*warning*)

The user-supplied derivatives of the objective function appear to be incorrect.

Large errors were found in the derivatives of the objective function. This value of **ifail** will occur if the verification process indicated that at least one gradient element had no correct figures. You should refer to the printed output to determine which elements are suspected to be in error.

As a first step, you should check that the code for the objective values is correct – for example, by computing the function at a point where the correct value is known. However, care should be taken that the chosen point fully tests the evaluation of the function. It is remarkable how often the values $x = 0$ or $x = 1$ are used to test function evaluation procedures, and how often the special properties of these numbers make the test meaningless.

Special care should be used in this test if computation of the objective function involves subsidiary data communicated in global storage. Although the first evaluation of the function may be correct, subsequent calculations may be in error because some of the subsidiary data has accidentally been overwritten.

Errors in programming the function may be quite subtle in that the function value is almost correct. For example, the function may not be accurate to full precision because of the inaccurate calculation of a subsidiary quantity, or the limited accuracy of data upon which the function depends. A common error on machines where numerical calculations are usually performed in double precision is to include even one single precision constant in the calculation of the function; since some compilers do not convert such constants to double precision, half the correct figures may be lost by such a seemingly trivial error.

ifail $= 8$

The gradient $\left(g = \dfrac{\partial F}{\partial x}\right)$ at the starting point x_0 is 'too small'. More precisely, the value of $g(x_0)^{\mathrm{T}} g(x_0)$ is less than $\epsilon_r |1 + F(x_0)|$, where ϵ_r is the value of the optional parameter **Function Precision** (default value $= \epsilon^{0.9}$).

The problem should be rerun from a different starting point.

ifail $= 9$

An input argument is invalid.

ifail $= -99$

An unexpected error has been triggered by this routine. Please contact NAG.

ifail $= -399$

Your licence key may have expired or may not have been installed correctly.

ifail $= -999$

Dynamic memory allocation failed.

7 Accuracy

On successful exit (**ifail** $= 0$) the accuracy of the solution will be as defined by the optional parameter **Optimality Tolerance** (default value $= \epsilon^{0.8}$).

8 Further Comments

To evaluate an 'acceptable' set of finite difference intervals using nag_opt_estimate_deriv (e04xa) requires 2 function evaluations per variable for a well-scaled problem and up to 6 function evaluations per variable for a badly scaled problem.

8.1 Description of Printed Output

This section describes the intermediate printout and final printout produced by nag_opt_uncon_conjgrd_comp (e04dg). You can control the level of printed output (see the description of the optional parameter **Print Level**). Note that the intermediate printout and final printout are produced only if **Print Level** ≥ 10 (the default).

The following line of summary output (< 80 characters) is produced at every iteration. In all cases, the values of the quantities are those in effect *on completion* of the given iteration.

`Itn` is the iteration count.

`Step` is the step α_k taken along the computed search direction. On reasonably well-behaved problems, the unit step (i.e., $\alpha_k = 1$) will be taken as the solution is approached.

`Nfun` is the cumulated number of evaluations of the objective function needed for the linesearch. Evaluations needed for the verification of the gradients by finite differences are not included. `Nfun` is printed as a guide to the amount of work required for the linesearch. nag_opt_uncon_conjgrd_comp (e04dg) will perform at most 11 function evaluations per iteration.

`Objective` is the value of the objective function at x_k.

`Norm G` is the Euclidean norm of the gradient of the objective function at x_k.

`Norm X` is the Euclidean norm of x_k.

`Norm (X(k-1)-X(k))` is the Euclidean norm of $x_{k-1} - x_k$.

The following describes the printout for each variable.

`Variable` gives the name (`Varbl`) and index j, for $j = 1, 2, \ldots, n$ of the variable.

`Value` is the value of the variable at the final iteration.

`Gradient Value` is the value of the gradient of the objective function with respect to the jth variable at the final iteration.

Numerical values are output with a fixed number of digits; they are not guaranteed to be accurate to this precision.

9 Example

This example finds a minimum of the function

$$F = e^{x_1}\left(4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1\right).$$

The initial point is

$$x_0 = (-1.0, 1.0)^{\mathrm{T}},$$

and $F(x_0) = 1.8394$ (to five figures).

The optimal solution is

$$x^* = (0.5, -1.0)^{\mathrm{T}},$$

and $F(x^*) = 0$.

9.1 Program Text

```
function e04dg_example

fprintf('e04dg example results\n\n');

x = [-1  1];

% Initialize
```

```
[cwsav,lwsav,iwsav,rwsav,ifail] = e04wb( ...
                                        'e04dg');

% Optimize
[iter, objf, objgrd, x, user, lwsav, iwsav, rwsav, ifail] = ...
e04dg( ...
       @objfun, x, lwsav, iwsav, rwsav);

fprintf('Variable          Value     Gradient value\n');
for i=1:2
 fprintf('Varbl %3d    %12.8f    %9.1e\n', i, x(i), objgrd(i));
end
fprintf('\nFinal objective value = %15.7e\n',objf);

function [mode, objf, objgrd, user] = objfun(mode, n, x, nstate, user)

  expx1 = exp(x(1));
  objf = expx1*(4*x(1)^2+2*x(2)^2+4*x(1)*x(2)+2*x(2)+1);
  if (mode == 2)
    objgrd(1) = 4*expx1*(2*x(1)+x(2)) + objf;
    objgrd(2) = 2*expx1*(2*x(2)+2*x(1)+1);
  else
    objgrd = zeros(2,1);
  end
```

9.2 Program Results

```
     e04dg example results

Variable          Value     Gradient value
Varbl   1      0.50000001      9.1e-07
Varbl   2     -0.99999989      8.3e-07

Final objective value =   5.3083002e-14
```

Note: *the remainder of this document is intended for more advanced users. Section 11 contains a detailed description of the algorithm which may be needed in order to understand Section 12. Section 12 describes the optional parameters which may be set by calls to nag_opt_uncon_conjgrd_option_string (e04dk).*

10 Algorithmic Details

This section contains a description of the method used by nag_opt_uncon_conjgrd_comp (e04dg).

nag_opt_uncon_conjgrd_comp (e04dg) uses a pre-conditioned conjugate gradient method and is based upon algorithm PLMA as described in Section 4.8.3 of Gill and Murray (1979) and Gill *et al.* (1981).

The algorithm proceeds as follows:

Let x_0 be a given starting point and let k denote the current iteration, starting with $k = 0$. The iteration requires g_k, the gradient vector evaluated at x_k, the kth estimate of the minimum. At each iteration a vector p_k (known as the direction of search) is computed and the new estimate x_{k+1} is given by $x_k + \alpha_k p_k$ where α_k (the step length) minimizes the function $F(x_k + \alpha_k p_k)$ with respect to the scalar α_k. A choice of initial step α_0 is taken as

$$\alpha_0 = \min\left\{1, 2 \times |F_k - F_{\text{est}}| / g_k^{\mathrm{T}} g_k\right\}$$

where F_{est} is a user-supplied estimate of the function value at the solution. If F_{est} is not specified, the software always chooses the unit step length for α_0. Subsequent step length estimates are computed using cubic interpolation with safeguards.

A quasi-Newton method can be used to compute the search direction p_k by updating the inverse of the approximate Hessian (H_k) and computing

$$p_{k+1} = -H_{k+1} g_{k+1}. \tag{1}$$

The updating formula for the approximate inverse is given by

$$H_{k+1} = H_k - \frac{1}{y_k^{\mathrm{T}} s_k}\left(H_k y_k s_k^{\mathrm{T}} + s_k y_k^{\mathrm{T}} H_k\right) + \frac{1}{y_k^{\mathrm{T}} s_k}\left(1 + \frac{y_k^{\mathrm{T}} H_k y_k}{y_k^{\mathrm{T}} s_k}\right) s_k s_k^{\mathrm{T}}, \tag{2}$$

where $y_k = g_{k-1} - g_k$ and $s_k = x_{k+1} - x_k = \alpha_k p_k$.

The method used to obtain the search direction is based upon computing p_{k+1} as $-H_{k+1} g_{k+1}$ where H_{k+1} is a matrix obtained by updating the identity matrix with a limited number of quasi-Newton corrections. The storage of an n by n matrix is avoided by storing only the vectors that define the rank two corrections – hence the term 'limited-memory' quasi-Newton method. The precise method depends upon the number of updating vectors stored. For example, the direction obtained with the 'one-step' limited memory update is given by (1) using (2) with H_k equal to the identity matrix, viz.

$$p_{k+1} = -g_{k+1} + \frac{1}{y_k^{\mathrm{T}} s_k}\left(s_k^{\mathrm{T}} g_{k+1} y_k + y_k^{\mathrm{T}} g_{k+1} s_k\right) - \frac{s_k^{\mathrm{T}} g_{k+1}}{y_k^{\mathrm{T}} s_k}\left(1 + \frac{y_k^{\mathrm{T}} y_k}{y_k^{\mathrm{T}} s_k}\right) s_k.$$

Using a limited-memory quasi-Newton formula, such as the one above, guarantees p_{k+1} to be a descent direction if all the inner products $y_k^{\mathrm{T}} s_k$ are positive for all vectors y_k and s_k used in the updating formula.

11 Optional Parameters

Several optional parameters in nag_opt_uncon_conjgrd_comp (e04dg) define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of nag_opt_uncon_conjgrd_comp (e04dg) these optional parameters have associated *default values* that are appropriate for most problems. Therefore, you need only specify those optional parameters whose values are to be different from their default values.

The remainder of this section can be skipped if you wish to use the default values for all optional parameters.

The following is a list of the optional parameters available. A full description of each optional parameter is provided in Section 12.1.

Defaults
Estimated Optimal Function Value
Function Precision
Iteration Limit
Iters
Itns
Linesearch Tolerance
List
Maximum Step Length
Nolist
Optimality Tolerance
Print Level
Start Objective Check at Variable
Stop Objective Check at Variable
Verify
Verify Gradients
Verify Level
Verify Objective Gradients

Optional parameters may be specified by calling nag_opt_uncon_conjgrd_option_string (e04dk) before a call to nag_opt_uncon_conjgrd_comp (e04dg).

nag_opt_uncon_conjgrd_option_string (e04dk) can be called to supply options directly, one call being necessary for each optional parameter. For example,

```
[lwsav, iwsav, rwsav, inform] = e04dk('Print Level = 1', lwsav, iwsav,
rwsav);
```

nag_opt_uncon_conjgrd_option_string (e04dk) should be consulted for a full description of this method of supplying optional parameters.

All optional parameters not specified by you are set to their default values. Optional parameters specified by you are unaltered by nag_opt_uncon_conjgrd_comp (e04dg) (unless they define invalid values) and so remain in effect for subsequent calls unless altered by you.

11.1 Description of the Optional Parameters

For each option, we give a summary line, a description of the optional parameter and details of constraints.

The summary line contains:

the keywords, where the minimum abbreviation of each keyword is underlined (if no characters of an optional qualifier are underlined, the qualifier may be omitted);

a parameter value, where the letters a, i and r denote options that take character, integer and real values respectively;

the default value, where the symbol ϵ is a generic notation for ***machine precision*** (see nag_machine_precision (x02aj)), and ϵ_r denotes the relative precision of the objective function **Function Precision**.

Keywords and character values are case and white space insensitive.

Defaults

This special keyword may be used to reset all optional parameters to their default values.

Estimated Optimal Function Value r

This value of r specifies the user-supplied guess of the optimum objective function value F_{est}. This value is used to calculate an initial step length α_0 (see Section 11). If the value of r is not specified (the default), then this has the effect of setting α_0 to unity. It should be noted that for badly scaled functions a unit step along the steepest descent direction will often compute the objective function at very large values of x.

Function Precision r Default $= \epsilon^{0.9}$

The parameter defines ϵ_r, which is intended to be a measure of the accuracy with which the problem function $F(x)$ can be computed. If $r < \epsilon$ or $r \geq 1$, the default value is used.

The value of ϵ_r should reflect the relative precision of $1 + |F(x)|$; i.e., ϵ_r acts as a relative precision when $|F|$ is large, and as an absolute precision when $|F|$ is small. For example, if $F(x)$ is typically of order 1000 and the first six significant digits are known to be correct, an appropriate value for ϵ_r would be 10^{-6}. In contrast, if $F(x)$ is typically of order 10^{-4} and the first six significant digits are known to be correct, an appropriate value for ϵ_r would be 10^{-10}. The choice of ϵ_r can be quite complicated for badly scaled problems; see Chapter 8 of Gill *et al.* (1981) for a discussion of scaling techniques. The default value is appropriate for most simple functions that are computed with full accuracy. However when the accuracy of the computed function values is known to be significantly worse than full precision, the value of ϵ_r should be large enough so that no attempt will be made to distinguish between function values that differ by less than the error inherent in the calculation.

Iteration Limit i Default $= \max(50, 5n)$
Iters
Itns

The value of i specifies the maximum number of iterations allowed before termination. If $i < 0$, the default value is used.

Problems whose Hessian matrices at the solution contain sets of clustered eigenvalues are likely to be minimized in significantly fewer than n iterations. Problems without this property may require anything between n and $5n$ iterations, with approximately $2n$ iterations being a common figure for moderately difficult problems.

Linesearch Tolerance r Default $= 0.9$

The value r controls the accuracy with which the step α taken during each iteration approximates a minimum of the function along the search direction (the smaller the value of r, the more accurate the linesearch). The default value $r = 0.9$ requests an inaccurate search, and is appropriate for most problems. A more accurate search may be appropriate when it is desirable to reduce the number of iterations – for example, if the objective function is cheap to evaluate. If $r < 0$ or $r \geq 1$, the default value is used.

List
Nolist Default for nag_opt_uncon_conjgrd_comp (e04dg)
= Nolist

Normally each optional parameter specification is printed as it is supplied. Optional parameter **Nolist** may be used to suppress the printing and optional parameter **List** may be used to restore printing.

Maximum Step Length r Default $= 10^{20}$

If $r > 0$, the maximum allowable step length for the linesearch is taken as $\min\left(\frac{1}{\text{x02am()}}, \frac{r}{\|p_k\|}\right)$. If $r \leq 0$, the default value is used.

Optimality Tolerance r Default $= \epsilon_R^{0.8}$

The parameter r specifies the accuracy to which you wish the final iterate to approximate a solution of the problem. Broadly speaking, r indicates the number of correct figures desired in the objective function at the solution. For example, if r is 10^{-6} and termination occurs with **ifail** $= 0$ (see Section 5), then the final point satisfies the termination criteria, where τ_F represents **Optimality Tolerance**. If $r < \epsilon_r$ or $r \geq 1$, the default value is used.

Print Level i

The value i controls the amount of printout produced by nag_opt_uncon_conjgrd_comp (e04dg), as indicated below. A detailed description of the printout is given in Section 9.1 (summary output at each iteration and the final solution).

i	**Output**
0	No output.
1	The final solution only.
5	One line of summary output (< 80 characters; see Section 9.1) for each iteration (no printout of the final solution).
10	The final solution and one line of summary output for each iteration.

Start Objective Check at Variable i_1 Default $= 1$
Stop Objective Check at Variable i_2 Default $= n$

These keywords take effect only if **Verify Level** > 0. They may be used to control the verification of gradient elements computed by **objfun**. For example, if the first 30 elements of the objective gradient appeared to be correct in an earlier run, so that only element 31 remains questionable, it is reasonable to

specify **Start Objective Check at Variable** $= 31$. If the first 30 variables appear linearly in the objective, so that the corresponding gradient elements are constant, the above choice would also be appropriate.

If $i_1 \leq 0$ or $i_1 > \max(1, \min(n, i_2))$, the default value is used. If $i_2 \leq 0$ or $i_2 > n$, the default value is used.

Verify Level i Default $= 0$
Verify
Verify Gradients
Verify Objective Gradients

These keywords refer to finite difference checks on the gradient elements computed by **objfun**. Gradients are verified at the user-supplied initial estimate of the solution. The possible choices for i are as follows:

i	**Meaning**
-1	No checks are performed.
0	Only a 'cheap' test will be performed, requiring one call to **objfun**.
1	In addition to the 'cheap' test, individual gradient elements will also be checked using a reliable (but more expensive) test.

For example, the objective gradient will be verified if **Verify**, **Verify** $=$ YES, **Verify Gradients**, **Verify Objective Gradients** or **Verify Level** $= 1$ is specified.