NAG Library Function Document

nag_opt_conj_grad (e04dgc)

1 Purpose

nag_opt_conj_grad (e04dgc) minimizes an unconstrained nonlinear function of several variables using a pre-conditioned, limited memory quasi-Newton conjugate gradient method. The function is intended for use on large scale problems.

2 Specification

```c
#include <nag.h>
#include <nage04.h>
void nag_opt_conj_grad (Integer n,
void (*objfun)(Integer n, const double x[], double *objf, double g[],
Nag_Comm *comm),
double x[], double *objf, double g[], Nag_E04_Opt *options,
Nag_Comm *comm, NagError *fail)
```

3 Description

nag_opt_conj_grad (e04dgc) uses a pre-conditioned conjugate gradient method and is based upon algorithm PLMA as described in Gill and Murray (1979) and Section 4.8.3 of 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 \( k \)th 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 \( \alpha_k \) (the step length) minimizes the function \( F(x_k + \alpha_k p_k) \) with respect to the scalar \( \alpha_k \).

At the start of each line search an initial approximation \( \alpha_0 \) to the step \( \alpha_k \) is taken as:

\[
\alpha_0 = \min \{1, 2|F_k - F_{est}|/g_k^T g_k\}
\]

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 \( \alpha_0 \). Subsequent step length estimates are computed using cubic interpolation with safeguards.

A quasi-Newton method computes the search direction, \( p_k \), by updating the inverse of the approximate Hessian \( H_k \) and computing

\[
p_{k+1} = -H_{k+1} g_{k+1}.
\]

The updating formula for the approximate inverse is given by

\[
H_{k+1} = H_k - \frac{1}{y_k^T s_k} (H_k y_k s_k^T + s_k y_k^T H_k) + \frac{1}{y_k^T s_k} \left(1 + \frac{y_k^T H_k y_k}{y_k^T s_k}ight) s_k s_k^T
\]

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

The method used by nag_opt_conj_grad (e04dgc) 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.
\[ p_{k+1} = -g_{k+1} + \frac{1}{y_k^Ts_k}(s_{k+1}^Tg_{k+1}y_k + y_{k+1}^Tg_{k+1}s_k) - \frac{s_{k+1}^Tg_{k+1}}{y_k^Ts_k}(1 + \frac{y_k^Ty_k}{y_k^Ts_k})s_k \]

nag_opt_conj_grad (e04dgc) uses a two-step method described in detail in Gill and Murray (1979) in which restarts and pre-conditioning are incorporated. 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 \( y_k^Ts_k \) are positive for all vectors \( y_k \) and \( s_k \) used in the updating formula.

The termination criteria of nag_opt_conj_grad (e04dgc) are as follows:

Let \( \tau_F \) specify an argument that indicates the number of correct figures desired in \( F_k \) (\( \tau_F \) is equivalent to \texttt{options.optim_tol} in the optional argument list, see Section 11). 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) \( \|g_k\| \leq \tau_F^{1/2}(1 + |F_k|) \) or \( \|g_k\| < \epsilon_A \), where \( \epsilon_A \) is the absolute error associated with computing the objective function

then the algorithm is considered to have converged. For a full discussion on termination criteria see Chapter 8 of Gill et al. (1981).

4 References


5 Arguments

1: \( n \) – Integer \hspace{1cm} Input

\textit{On entry:} the number \( n \) of variables.

\textit{Constraint:} \( n \geq 1 \).

2: \texttt{objfun} – function, supplied by the user \hspace{1cm} External Function

\texttt{objfun} must calculate the objective function \( F(x) \) and its gradient at a specified point \( x \).

The specification of \texttt{objfun} is:

\begin{verbatim}
void objfun (Integer n, const double x[], double *objf, double g[],
Nag_Comm *comm)
1: \( n \) – Integer \hspace{1cm} Input

\textit{On entry:} the number \( n \) of variables.

2: \( x[n] \) – const double \hspace{1cm} Input

\textit{On entry:} the point \( x \) at which the objective function is required.

3: \texttt{objf} – double * \hspace{1cm} Output

\textit{On exit:} the value of the objective function \( F \) at the current point \( x \).
\end{verbatim}
On exit: $g[i-1]$ must contain the value of $\frac{\partial F}{\partial x_i}$ at the point $x_i$ for $i = 1, 2, \ldots, n$.

**Note:** $\text{objfun}$ should be tested separately before being used in conjunction with nag_opt_conj_grad (e04dgc). The array $x$ must not be changed by $\text{objfun}$. 

On entry: $x_0$, an estimate of the solution point $x^*$. 

On exit: the final estimate of the solution.

On exit: the value of the objective function $F(x)$ at the final iterate.

On exit: the objective gradient at the final iterate.

On entry/exit: a pointer to a structure of type Nag_E04_Opt whose members are optional arguments for nag_opt_conj_grad (e04dgc). These structure members offer the means of adjusting some of the argument values of the algorithm and on output will supply further details of the results. A description of the members of options is given below in Section 11.

If any of these optional arguments are required then the structure options should be declared and initialized by a call to nag_opt_init (e04xxc) and supplied as an argument to nag_opt_conj_grad (e04dgc). However, if the optional arguments are not required the NAG defined null pointer, E04_DEFAULT, can be used in the function call.
7: comm – Nag_Comm *

Input/Output

Note: comm is a NAG defined type (see Section 3.2.1.1 in the Essential Introduction).

On entry/exit: structure containing pointers for communication with user-supplied functions; see
the above description of objfun for details. If you do not need to make use of this communication
feature the null pointer NAGCOMM_NULL may be used in the call to nag_opt_conj_grad (e04dgc);
comm will then be declared internally for use in calls to user-supplied functions.

8: fail – NagError *

Input/Output

The NAG error argument (see Section 3.6 in the Essential Introduction).

5.1 Description of Printed Output

Intermediate and final results are printed out by default. The level of printed output can be controlled
with the structure member options.print_level (see Section 11.2). The default,
options.print_level = Nag_Soln_Iter, provides the result of any derivative check, a single line of output
at each iteration and the final result.

The derivative check performed by default will give the directional derivative, $g(x)^T p$, of the objective
gradient and its finite difference approximation, where $p$ is a random vector of unit length. If the gradient
is believed to be in error then nag_opt_conj_grad (e04dgc) will exit with fail.code = NE_DERIV_ERRORS.

The line of results printed at each iteration gives:

- **Itn**: the current iteration number $k$.
- **Nfun**: the cumulative number of calls to objfun. The evaluations needed for the
  estimation of the gradients by finite differences are not included in the total Nfun.
  The value of Nfun is a guide to the amount of work required for the linesearch.
  nag_opt_conj_grad (e04dgc) will perform at most 16 function evaluations per
  iteration.
- **Objective**: the current value of the objective function, $F(x_k)$.
- **Norm g**: the Euclidean norm of the gradient vector, $\|g(x_k)\|$.
- **Norm x**: the Euclidean norm of $x_k$.
- **Norm (x(k-1)-x(k))**: the Euclidean norm of $x_{k-1} - x_k$.
- **Step**: the step $\alpha_k$ taken along the computed search direction $p_k$. On reasonably well-
  behaved problems, the unit step will be taken as the solution is approached.

The printout of the final result consists of:

- **x**: the final point, $x^*$.
- **g**: the final gradient vector, $g(x^*)$.

6 Error Indicators and Warnings

**NE_ALLOC_FAIL**

Dynamic memory allocation failed.

**NE_BAD_PARAM**

On entry, argument options.print_level had an illegal value.

On entry, argument options.verify_grad had an illegal value.
**NE_DERIV_ERRORS**

Large errors were found in the derivatives of the objective function.
This value of fail will occur if the verification process indicated that at least one gradient component 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.

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.

**NE_GRAD_TOO_SMALL**

The gradient at the starting point is too small, rerun the problem at a different starting point.

The value of $g(x_0)^T g(x_0)$ is less than $\epsilon |F(x_0)|$, where $\epsilon$ is the *machine precision*.

**NE_INT_ARG_LT**

On entry, $n = (value)$. Constraint: $n \geq 1$.

**NE_INVALID_INT_RANGE_1**

Value $(value)$ given to options.max_iter not valid. Correct range is options.max_iter $\geq 0$.

**NE_INVALID_REAL_RANGE_EF**

Value $(value)$ given to options.f_prec not valid. Correct range is $\epsilon \leq$ options.f_prec $< 1.0$.

Value $(value)$ given to options.optim_tol not valid. Correct range is $(value) \leq$ options.optim_tol $< 1.0$.

**NE_INVALID_REAL_RANGE_F**

Value $(value)$ given to options.max_line_step not valid. Correct range is options.max_line_step $> 0.0$.

**NE_INVALID_REAL_RANGE_FF**

Value $(value)$ given to options.linesearch_tol not valid. Correct range is $0.0 \leq$ options.linesearch_tol $< 1.0$.

**NE_NOT_APPEND_FILE**

Cannot open file $(string)$ for appending.

**NE_NOT_CLOSE_FILE**

Cannot close file $(string)$.

**NE_OPT_NOT_INIT**

Options structure not initialized.

**NE_USER_STOP**

User requested termination, user flag value = $(value)$.
This exit occurs if you set `comm->flag` to a negative value in `objfun`. If `fail` is supplied the value of `fail.errnum` will be the same as your setting of `comm->flag`.

**NE_WRITE_ERROR**

Error occurred when writing to file `(string)`.

**NW_NO_IMPROVEMENT**

A sufficient decrease in the function value could not be attained during the final linesearch. Current point cannot be improved upon.

If `objfun` computes the function and gradients correctly, then this warning may occur because an overly stringent accuracy has been requested, i.e., `options.optim_tol` is too small or if the minimum lies close to a step length of zero. In this case you should apply the tests described in Section 3 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_conj_grad (e04dgc) has failed to move from the initial point, then the function `objfun` may be incorrect. You should refer to the comments below under NE_DERIV_ERRORS and check the gradients using the `options.verify_grad` argument. 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.

**NW_STEP_BOUND_TOO_SMALL**

Computed upper-bound on step length was too small

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

**NW_TOO_MANY_ITER**

The maximum number of iterations, `(value)`, have been performed.

If the algorithm appears to be making progress the value of `options.max_iter` value may be too small (see Section 11), you should increase its value and rerun nag_opt_conj_grad (e04dgc). If the algorithm seems to be ‘bogged down’, you should check for incorrect gradients or ill-conditioning as described below under NW_NO_IMPROVEMENT.

7 Accuracy

On successful exit the accuracy of the solution will be as defined by the optional argument `options.optim_tol`.

8 Parallelism and Performance

Not applicable.

9 Further Comments

9.1 Timing

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.
10  Example

This example minimizes the function

\[ F = e^x (4x_1^2 + 2x_2^2 + 4x_1x_2 + 2x_2 + 1) \]

The data includes a set of user-defined column and row names, and data for the Hessian in a sparse storage format (see Section 10.2 for further details). The options structure is declared and initialized by nag_opt_init (e04xxc). Five option values are read from a data file by use of nag_opt_read (e04xyc).

10.1  Program Text

```c
/* nag_opt_conj_grad (e04dgc) Example Program. *
 * Copyright 2014 Numerical Algorithms Group. *
 * Mark 2, 1991. *
 * Mark 7 revised, 2001. *
 * Mark 8 revised, 2004. *
 */
#include <nag.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <nag_stdlib.h>
#ifdef __cplusplus
extern "C" {
#endif
static void NAG_CALL objfun(Integer n, const double x[], double *objf, double g[], Nag_Comm *comm);
#ifdef __cplusplus
}
#endif
int main(void)
{
    const char *optionsfile = "e04dgce.opt";
    static double ruser[1] = {-1.0};
    Integer exit_status = 0;
    Nag_Boolean print;
    Integer n;
    Nag_E04_Opt options;
    double *g = 0, objf, *x = 0;
    Nag_Comm comm;
    NagError fail;
    INIT_FAIL(fail);

    printf("nag_opt_conj_grad (e04dgc) Example Program Results\n");

    /* For communication with user-supplied functions: */
    comm.user = ruser;
    fflush(stdout);

    /* Initialise options structure and read option values from file */
    print = Nag_TRUE;
    n = 2; /* Number of variables */
    if (n >= 1)
    {
        if (!(*x = NAG_ALLOC(n, double))) ||
            (!g = NAG_ALLOC(n, double))
        {
            printf("Allocation failure\n");
        }
    }
    return 0;
}
```

exit_status = -1;
goto END;
}
else
{
    printf("Invalid n.\n");
    exit_status = 1;
    return exit_status;
}

/* nag_opt_init (e04xxc).
 * Initialization function for option setting
 */
nag_opt_init(&options);

/* nag_opt_read (e04xyc).
 * Read options from a text file
 */
nag_opt_read("e04dgc", optionsfile, &options, print, "stdout", &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_opt_read (e04xyc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

/* Set the initial estimate of the solution. */
x[0] = -1.0;
x[1] = 1.0;

/* Solve the problem. */
/* nag_opt_conj_grad (e04dgc), see above. */
nag_opt_conj_grad(n, objfun, x, &objf, g, &options, &comm, &fail);
if (fail.code != NE_NOERROR)
{
    printf("Error from nag_opt_conj_grad (e04dgc).\n%s\n", fail.message);
    exit_status = 1;
    goto END;
}

END:
NAG_FREE(x);
NAG_FREE(g);

return exit_status;
}

static void NAG_CALL objfun(Integer n, const double x[], double *objf,
                             double g[], Nag_Comm *comm)
{
    /* Function to evaluate objective function and its 1st derivatives. */

double ex1, x1, x2;
if (comm->user[0] == -1.0)
{
    printf("(User-supplied callback objfun, first invocation.)\n");
    fflush(stdout);
    comm->user[0] = 0.0;
}

ex1 = exp(x[0]);
x1 = x[0];
x2 = x[1];
*objf = ex1*(4*x1*x1 + 2*x2*x2 + 4*x1*x2 + 2*x2 + 1);
g[0] = 4*ex1*(2*x1 + x2) + *objf;
g[1] = 2*ex1*(2*x2 + 2*x1 + 1);
}

/* objfun */
10.2 Program Data

nag_opt_conj_grad (e04dgc) Example Program Optional Parameters

Following options for e04dgc are read by e04xyc.

begin e04dgc
  print_level = Nag_Soln /* Print solution only */
  max_iter = 30 /* Set iteration limit */
  verify_grad = Nag_CheckObj /* Check objective gradient components */
  max_line_step = 1.0e+2 /* Maximum allowable step length */
  f_est = 1.0 /* Estimate of optimal function value */
end

10.3 Program Results

nag_opt_conj_grad (e04dgc) Example Program Results

Optional parameter setting for e04dgc.

--------------------------------------
Option file: e04dgce.opt
print_level set to Nag_Soln
max_iter set to 30
verify_grad set to Nag_CheckObj
max_line_step set to 1.00e+02
f_est set to 1.00e+00

Parameters to e04dgc
--------------------
Number of variables........... 2
max_line_step........... 1.00e+02 machine precision....... 1.11e-16
optim_tol............... 3.26e-12 linesearch_tol.......... 9.00e-01
f_est................... 1.00e+00 f_prec.................. 4.38e-15
verify_grad......... Nag_CheckObj max_iter............... 30
print_level......... Nag_Soln print_gcheck............ Nag_TRUE
outfile................. stdout

(User-supplied callback objfun, first invocation.)

Verification of the objective gradients.

All objective gradient elements have been set.
The objective gradient seems to be ok.

Directional derivative of the objective -1.47151776e-01
Difference approximation -1.47151796e-01

Component-wise check:

<table>
<thead>
<tr>
<th>i</th>
<th>x[i]</th>
<th>dx[i]</th>
<th>g[i]</th>
<th>Difference approxn.</th>
<th>Itns.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-1.00e+00</td>
<td>1.64e-07</td>
<td>3.67879441e-01</td>
<td>3.67879441e-01 OK</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1.00e+00</td>
<td>1.84e-07</td>
<td>7.35758882e-01</td>
<td>7.35758882e-01 OK</td>
<td>1</td>
</tr>
</tbody>
</table>

2 objective gradient elements out of the 2 assigned,
set in columns 1 through 2, seem to be ok.
The largest relative error was 5.08e-11 in element 2
Results from e04dgc:
-------------------
Final solution:

<table>
<thead>
<tr>
<th>Variable</th>
<th>x</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5.0000e-01</td>
<td>1.3247e-07</td>
</tr>
<tr>
<td>2</td>
<td>-1.0000e+00</td>
<td>3.0215e-08</td>
</tr>
</tbody>
</table>

Final objective function value = 8.6231277e-16.
Exit after 9 iterations and 19 function evaluations.

Optimal solution found.

11 Optional Arguments

A number of optional input and output arguments to nag_opt_conj_grad (e04dgc) are available through the structure argument options, type Nag_E04_Opt. An argument may be selected by assigning an appropriate value to the relevant structure member; those arguments not selected will be assigned default values. If no use is to be made of any of the optional arguments you should use the NAG defined null pointer, E04_DEFAULT, in place of options when calling nag_opt_conj_grad (e04dgc); the default settings will then be used for all arguments.

Before assigning values to options directly the structure must be initialized by a call to the function nag_opt_init (e04xxc). Values may then be assigned to the structure members in the normal C manner.

Option settings may also be read from a text file using the function nag_opt_read (e04xyc) in which case initialization of the options structure will be performed automatically if not already done. Any subsequent direct assignment to the options structure must not be preceded by initialization.

If assignment of functions and memory to pointers in the options structure is required, then this must be done directly in the calling program, they cannot be assigned using nag_opt_read (e04xyc).

11.1 Optional Argument Checklist and Default Values

For easy reference, the following list shows the members of options which are valid for nag_opt_conj_grad (e04dgc) together with their default values where relevant. The number $\epsilon$ is a generic notation for machine precision (see nag_machine_precision (X02AJC)).

<table>
<thead>
<tr>
<th>Boolean list</th>
<th>Nag_TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nag_PrintType print_level</td>
<td>Nag_Soln_Iter</td>
</tr>
<tr>
<td>char outfile[80]</td>
<td>stdout</td>
</tr>
<tr>
<td>void (*print_fun)()</td>
<td>NULL</td>
</tr>
<tr>
<td>Nag_GradChk verify_grad</td>
<td>Nag_SimpleCheck</td>
</tr>
<tr>
<td>Boolean print_gcheck</td>
<td>Nag_TRUE</td>
</tr>
<tr>
<td>Integer obj_check_start</td>
<td>1</td>
</tr>
<tr>
<td>Integer obj_check_stop</td>
<td>n</td>
</tr>
<tr>
<td>Integer max_iter</td>
<td>max(50, 5n)</td>
</tr>
<tr>
<td>double f_prec</td>
<td>$\epsilon^{0.9}$</td>
</tr>
<tr>
<td>double optim_tol</td>
<td>options.f_prec$^{0.8}$</td>
</tr>
<tr>
<td>double linesearch_tol</td>
<td>0.9</td>
</tr>
<tr>
<td>double max_line_step</td>
<td>$10^{10}$</td>
</tr>
<tr>
<td>double f_est</td>
<td>iter</td>
</tr>
<tr>
<td>Integer nf</td>
<td>nf</td>
</tr>
</tbody>
</table>
11.2 Description of the Optional Arguments

list – Nag_Boolean

Default = Nag_TRUE

On entry: if options.list = Nag_TRUE the argument settings in the call to nag_opt_conj_grad (e04dgc) will be printed.

print_level – Nag_PrintType

Default = Nag_Soln.Iter

On entry: the level of results printout produced by nag_opt_conj_grad (e04dgc). The following values are available:

Nag_NoPrint No output.
Nag_Soln The final solution.
Nag_Iter One line of output for each iteration.
Nag_Soln_Iter The final solution and one line of output for each iteration.

Constraint: options.print_level = Nag_NoPrint, Nag_Soln, Nag_Iter or Nag_Soln_Iter.

outfile – const char[80]

Default = stdout

On entry: the name of the file to which results should be printed. If options.outfile[0] = ‘\0’ then the stdout stream is used.

print_fun – pointer to function

Default = NULL

On entry: printing function defined by you; the prototype of options.print_fun is

void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);

See Section 11.3.1 below for further details.

verify_grad – Nag_GradChk

Default = Nag_SimpleCheck

On entry: specifies the level of derivative checking to be performed by nag_opt_conj_grad (e04dgc) on the gradient elements defined in objfun.

options.verify_grad may have the following values:

Nag_NoCheck No derivative check is performed.
Nag_SimpleCheck Perform a simple check of the gradient.
Nag_CheckObj Perform a component check of the gradient elements.

If options.verify_grad = Nag_SimpleCheck then a simple ‘cheap’ test is performed, which requires only one call to objfun. If options.verify_grad = Nag_CheckObj then a more reliable (but more expensive) test will be made on individual gradient components. This component check will be made in the range specified by options.obj_check_start and options.obj_check_stop, default values being 1 and n respectively. The procedure for the derivative check is based on finding an interval that produces an acceptable estimate of the second derivative, and then using that estimate to compute an interval that should produce a reasonable forward-difference approximation. The gradient element is then compared with the difference approximation. (The method of finite difference interval estimation is based on Gill et al. (1983)). The result of the test is printed out by nag_opt_conj_grad (e04dgc) if options.print_gcheck = Nag_TRUE.

Constraint: options.verify_grad = Nag_NoCheck, Nag_SimpleCheck or Nag_CheckObj.

print_gcheck – Nag_Boolean

Default = Nag_TRUE

On entry: if Nag_TRUE the result of any derivative check (see options.verify_grad) will be printed.
On entry: these options take effect only when `options.verify_grad` = Nag_CheckObj. They may be used to control the verification of gradient elements computed by the function `objfun`. For example, if the first 30 variables appear linearly in the objective, so that the corresponding gradient elements are constant, then it is reasonable for `options.obj_check_start` to be set to 31.

Constraint: $1 \leq \text{options.obj_check_start} \leq \text{options.obj_check_stop} \leq n$.

On entry: the limit on the number of iterations allowed before termination.

Constraint: `options.max_iter` $\geq 0$.

On entry: this argument defines $\epsilon_r$, which is intended to be a measure of the accuracy with which the problem function $F$ can be computed. The value of $\epsilon_r$ should reflect the relative precision of $1 + |F(x)|$; i.e., $\epsilon_r$ 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 $\epsilon_r$ would be $1.0 \times 10^{-6}$. 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 $\epsilon_r$ would be $1.0 \times 10^{-10}$. The choice of $\epsilon_r$ 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 $\epsilon_r$ should be large enough so that `nag_opt_conj_grad` (e04dgc) will not attempt to distinguish between function values that differ by less than the error inherent in the calculation.

Constraint: $\epsilon \leq \text{options.f_prec} < 1.0$.

On entry: specifies the accuracy to which you wish the final iterate to approximate a solution of the problem. Broadly speaking, `options.optim_tol` indicates the number of correct figures desired in the objective function at the solution. For example, if `options.optim_tol` is $10^{-6}$ and `nag_opt_conj_grad` (e04dgc) terminates successfully, the final value of $F$ should have approximately six correct figures. `nag_opt_conj_grad` (e04dgc) will terminate successfully if the iterative sequence of $x$-values is judged to have converged and the final point satisfies the termination criteria (see Section 3, where $\tau_F$ represents `options.optim_tol`).

Constraint: $\text{options.f_prec} \leq \text{options.optim_tol} < 1.0$.

On entry: controls the accuracy with which the step $\alpha$ taken during each iteration approximates a minimum of the function along the search direction (the smaller the value of `options.linesearch_tol`, the more accurate the linesearch). The default value 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.

Constraint: $0.0 \leq \text{options.linesearch_tol} < 1.0$.

On entry: defines the maximum allowable step length for the line search.

Constraint: `options.max_line_step` $> 0.0$.

On entry: specifies the user-supplied guess of the optimum objective function value. This value is used by `nag_opt_conj_grad` (e04dgc) to calculate an initial step length (see Section 3). If no value is supplied...
then an initial step length of 1.0 will be used but it should be noted that for badly scaled functions a unit step along the steepest descent direction will often compute the function at very large values of $x$.

**iter** – Integer

*On exit:* the number of iterations which have been performed in nag_opt_conj_grad (e04dgc).

**nf** – Integer

*On exit:* the number of times the objective function has been evaluated (i.e., number of calls of **objfun**). The total excludes the calls made to **objfun** for purposes of derivative checking.

### 11.3 Description of Printed Output

The level of printed output can be controlled with the structure members **options.list**, **options.print_gcheck** and **options.print_level** (see Section 11.2). If **options.list** = Nag_TRUE then the argument values to nag_opt_conj_grad (e04dgc) are listed, followed by the result of any derivative check if **options.print_gcheck** = Nag_TRUE. The printout of the optimization results is governed by the value of **options.print_level**. The default of **options.print_level** = Nag_Soln_Iter provides a single line of output at each iteration and the final result. This section describes all of the possible levels of results printout available from nag_opt_conj_grad (e04dgc).

If a simple derivative check, **options.verify_grad** = Nag_SimpleCheck, is requested then the directional derivative, $g(x)^T p$, of the objective gradient and its finite difference approximation are printed out, where $p$ is a random vector of unit length.

When a component derivative check, **options.verify_grad** = Nag_CheckObj, is requested then the following results are supplied for each component:

- $x[i]$ the element of $x$.
- $dx[i]$ the optimal finite difference interval.
- $g[i]$ the gradient element.
- Difference approxn. the finite difference approximation.
- Itns the number of trials performed to find a suitable difference interval.

The indicator, OK or BAD?, states whether the gradient element and finite difference approximation are in agreement.

If the gradient is believed to be in error nag_opt_conj_grad (e04dgc) will exit with fail set to NE_DERIV_ERRORS.

When **options.print_level** = Nag_Iter or Nag_Soln_Iter a single line of output is produced on completion of each iteration, this gives the following values:

- Itn the current iteration number $k$.
- Nfun the cumulative number of calls to **objfun**. The evaluations needed for the estimation of the gradients by finite differences are not included in the total Nfun. The value of Nfun is a guide to the amount of work required for the linesearch. nag_opt_conj_grad (e04dgc) will perform at most 16 function evaluations per iteration.
- Objective the current value of the objective function, $F(x_k)$.
- Norm g the Euclidean norm of the gradient vector, $||g(x_k)||$.
- Norm x the Euclidean norm of $x_k$.
- Norm(x(k-1)-x(k)) the Euclidean norm of $x_{k-1} - x_k$.
- Step the step $\alpha$ taken along the computed search direction $p_k$. 

---

Mark 25
If `options.print_level = Nag_Soln` or `Nag_Soln_Iter`, the final result is printed out. This consists of:

- $x$ the final point, $x^*$.
- $g$ the final gradient vector, $g(x*)$.

If `options.print_level = Nag_NoPrint` then printout will be suppressed; you can print the final solution when nag_opt_conj_grad (e04dgc) returns to the calling program.

### 11.3.1 Output of results via a user-defined printing function

You may also specify your own print function for output of the results of any gradient check, the optimization results at each iteration and the final solution. The user-defined print function should be assigned to the `options.print_fun` function pointer, which has prototype

```c
void (*print_fun)(const Nag_Search_State *st, Nag_Comm *comm);
```

The rest of this section can be skipped if the default printing facilities provide the required functionality.

When a user-defined function is assigned to `options.print_fun` this will be called in preference to the internal print function of nag_opt_conj_grad (e04dgc). Calls to the user-defined function are again controlled by means of the `options.print_check` and `options.print_level` members. Information is provided through `st` and `comm` the two structure arguments to `options.print_fun`.

If `comm->it prt = Nag_TRUE` then the results from the last iteration of nag_opt_conj_grad (e04dgc) are in the following members of `st`:

- `n` – Integer
  The number of variables.
- `x` – double *
  Points to the `st->n` memory locations holding the current point $x_k$.
- `f` – double
  The value of the current objective function.
- `g` – double *
  Points to the `st->n` memory locations holding the first derivatives of $F$ at the current point $x_k$.
- `step` – double
  The step $\alpha$ taken along the search direction $p_k$.
- `xk_norm` – double
  The Euclidean norm of $x_{k-1} - x_k$.
- `iter` – Integer
  The number of iterations performed by nag_opt_conj_grad (e04dgc).
- `nf` – Integer
  The cumulative number of calls made to `objfun`.

If `comm->g prt = Nag_TRUE` then the following members are set:

- `n` – Integer
  The number of variables.
- `x` – double *
  Points to the `st->n` memory locations holding the initial point $x_0$.
- `g` – double *
  Points to the `st->n` memory locations holding the first derivatives of $F$ at the initial point $x_0$. 
Details of any derivative check performed by nag_opt_conj_grad (e04dgc) are held in the following substructure of st:

**gprint** – Nag_GPrintSt *

Which in turn contains two substructures gprint→g_chk, gprint→f_sim and a pointer to an array of substructures, *gprint→f_comp.

**g_chk** – Nag_Grad_Chk_St *

This substructure contains the members:

- **type** – Nag_GradChk
  The type of derivative check performed by nag_opt_conj_grad (e04dgc). This will be the same value as in options.verify_grad.

- **g_error** – Integer
  This member will be equal to one of the error codes NE_NOERROR or NE_DERIV_ERRORS according to whether the derivatives were found to be correct or not.

- **obj_start** – Integer
  Specifies the gradient element at which any component check started. This value will be equal to options.obj_check_start.

- **obj_stop** – Integer
  Specifies the gradient element at which any component check ended. This value will be equal to options.obj_check_end.

**f_sim** – Nag_SimSt *

The result of a simple derivative check, gchk→type = Nag_SimpleCheck, will be held in this substructure which has members:

- **correct** – Nag_Boolean
  If Nag_TRUE then the objective gradient is consistent with the finite difference approximation according to a simple check.

- **dir_deriv** – double *
  The directional derivative $g(x)^Tp$ where $p$ is a random vector of unit length with elements of approximately equal magnitude.

- **fd_approx** – double *
  The finite difference approximation, $(F(x + hp) - F(x))/h$, to the directional derivative.

**f_comp** – Nag_CompSt *

The results of a component derivative check, gchk→type = Nag_CheckObj, will be held in the array of st→n substructures of type Nag_CompSt pointed to by gprint→f_comp. The procedure for the derivative check is based on finding an interval that produces an acceptable estimate of the second derivative, and then using that estimate to compute an interval that should produce a reasonable forward-difference approximation. The gradient element is then compared with the difference approximation. (The method of finite difference interval estimation is based on Gill et al. (1983)).

- **correct** – Nag_Boolean
  If Nag_TRUE then this objective gradient component is consistent with its finite difference approximation.
hopt – double *

The optimal finite difference interval. This is $dx[i]$ in the NAG default printout.

gdiff – double *

The finite difference approximation for this gradient component.

iter – Integer

The number of trials performed to find a suitable difference interval.

comment – char

A character string which describes the possible nature of the reason for which an estimation of the finite difference interval failed to produce a satisfactory relative condition error of the second-order difference. Possible strings are: "Constant?", "Linear or odd?", "Too nonlinear?" and "Small derivative?".

The relevant members of the structure comm are:

g_prt – Nag_Boolean

Will be Nag_TRUE only when the print function is called with the result of the derivative check of objfun.

it_prt – Nag_Boolean

Will be Nag_TRUE when the print function is called with the result of the current iteration.

sol_prt – Nag_Boolean

Will be Nag_TRUE when the print function is called with the final result.

user – double *

iuser – Integer *

p – Pointer

Pointers for communication of user information. If used they must be allocated memory either before entry to nag_opt_conj_grad (e04dgc) or during a call to objfun or options.print_fun. The type Pointer will be void * with a C compiler that defines void * and char * otherwise.