NAG Library Function Document

nag_opt_handle_solve_dfls (e04ffc)

Note: *this function uses* **optional parameters** *to define choices in the problem specification and in the details of the algorithm. If you wish to use* default *settings for all of the optional parameters, you need only read Sections 1 to 10 of this document. If, however, you wish to reset some or all of the settings please refer to Section 11 for a detailed description of the algorithm and to Section 12 for a detailed description of the specification of the optional parameters*.

1 Purpose

nag_opt_handle_solve_dfls (e04ffc) is a derivative free solver from the NAG optimization modelling suite for small to medium-scale nonlinear least squares problems with bound constraints.

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_handle_solve_dfls (void *handle,
     void (*objfun)(Integer nvar, const double x[], Integer nres,
          double rx[], Integer *inform, Nag_Comm *comm),
     void (*mon)(Integer nvar, const double x[], Integer *inform,
          const double rinfo[], const double stats[], Nag_Comm *comm),
     Integer nvar, double x[], Integer nres, double rx[], double rinfo[],
     double stats[], Nag_Comm *comm, NagError *fail)
```

3 Description

nag_opt_handle_solve_dfls (e04ffc) serves as a solver for compatible problems stored as a handle. The handle points to an internal data structure which defines the problem and serves as a means of communication for functions in the suite.

nag_opt_handle_solve_dfls (e04ffc) is aimed at minimizing a sum of a squares objective function subject to bound constraints:

$$\begin{aligned} \underset{x\in R^n}{\text{minimize}} \quad & \sum_{i=1}^{m_r} r_i(x)^2 \qquad \text{(a)} \\ \text{subject to} \quad & l_x \leq x \leq u_x, \qquad \text{(b)} \end{aligned} \tag{1}$$

Here the $r_i(x)$ are smooth nonlinear functions called residuals and l_x and u_x are n-dimensional vectors defining bounds on the variables. Typically, in a calibration or data fitting context, the residuals will be defined as the difference between a data point and a nonlinear model (see Section 2.2.3 in the e04 Chapter Introduction)

To define a compatible problem handle, you must call **nag_opt_handle_init (e04rac)** followed by **nag_opt_handle_set_nlnls (e04rmc)** to initialize it and optionally call **nag_opt_handle_set_simple bounds (e04rhc)** to define bounds on the variables. If **nag_opt_handle_set_simplebounds (e04rhc)** is not called, all the variables will be considered free by the solver. It should be noted that **nag_opt_handle_solve_dfls (e04ffc)** always assumes that the Jacobian of the residuals is dense, therefore defining a sparse structure for the residuals in the call to **nag_opt_handle_set_nlnls (e04rmc)** will have no effect.

It is possible to fix some variables with the definition of the bounds. However, some constraints must be met in order to be able to call the solver:

the number of non-fixed variables n_r has to be at least 2

for all non-fixed variable x_i, the value of $u_x(i) - l_x(i)$ has to be at least twice the starting trust region radius (see the consistency constraint of the optional parameter **DFLS Starting Trust Region**).

The solver is based on a derivative free trust region framework. This type of method is well suited for small to medium-scale problems (around 100 variables) for which the derivatives are unavailable or not easy to compute and/or for which the function evaluations are expensive or noisy. For a detailed description of the algorithm see Section 11. The algorithm behaviour and solver strategy can be modified by various optional parameters (see Section 12) which can be set by **nag_opt_handle_opt_set (e04zmc)** and **nag_opt_handle_opt_set_file (e04zpc)** anytime between the initialization of the handle by **nag_opt_handle_init (e04rac)** and a call to the solver. The default values for these optional parameters are chosen to work well in the general case but it is recommended to tune them to your particular problem. In particular, if the objective function is noisy, it is highly recommended to set the optional parameter **DFLS Trust Region Update** to SLOW to improve convergence. Once the solver has finished, options may be modified for the next solve. The solver may be called repeatedly with various starting points and/or optional parameters.

4 References

Powell M J D (2009) The BOBYQA algorithm for bound constrained optimization without derivatives *Report DAMTP 2009/NA06* University of Cambridge http://www.damtp.cam.ac.uk/user/na/NA_papers/NA2009_06.pdf

Zhang H, CONN A R and Scheinberg k (2010) A Derivative-Free Algorithm for Least-Squares Minimization *SIAM J. Optim.* **20(6)** 3555–3576

5 Arguments

1: **handle** – void * *Input*

On entry: the handle to the problem. It needs to be initialized by **nag_opt_handle_init (e04rac)** and the objective must be declared as nonlinear least squares by a call to the function **nag_opt_handle_set_nlnls (e04rmc)**. The function **nag_opt_handle_set_simplebounds (e04rhc)** can optionally be called to define box bounds. It **must not** be changed between calls to the NAG optimization modelling suite.

2: **objfun** – function, supplied by the user *External Function*

objfun must evaluate the value of the nonlinear residuals $r_i(x)$ at a specified point x.

The specification of **objfun** is:

```
void objfun (Integer nvar, const double x[], Integer nres,
     double rx[], Integer *inform, Nag_Comm *comm)
```

1: **nvar** – Integer *Input*

On entry: n, the number of variables in the problem, as set during the initialization of the handle by **nag_opt_handle_init (e04rac)**.

2: **x**[**nvar**] – const double *Input*

On entry: x, the vector of variable values at which the residuals r_i are to be evaluated.

3: **nres** – Integer *Input*

On entry: m_r, the number of residuals in the problem, as set during the initialization of the handle by **nag_opt_handle_set_nlnls (e04rmc)**.

4: **rx**[**nres**] – double *Output*

On exit: the value of the residuals $r_i(x)$ at x.

5: **inform** – Integer * *Input/Output*

On entry: a non-negative value.

On exit: may be used to request the solver to stop immediately. Specifically, if **inform** < 0 then the value of **rx** will be discarded and the solver will terminate immediately with **fail**.**code** = NE_USER_STOP otherwise, the solver will proceed normally.

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **objfun**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling **nag_opt_handle_solve_dfls (e04ffc)** you may allocate memory and initialize these pointers with various quantities for use by **objfun** when called from **nag_opt_handle_solve_dfls (e04ffc)** (see Section 3.3.1.1 in How to Use the NAG Library and its Documentation).

Note: objfun should not return floating-point NaN (Not a Number) or infinity values, since these are not handled by **nag_opt_handle_solve_dfls (e04ffc)**. If your code inadvertently **does** return any NaNs or infinities, **nag_opt_handle_solve_dfls (e04ffc)** is likely to produce unexpected results.

3: **mon** – function, supplied by the user *External Function*

mon is provided to enable you to monitor the progress of the optimization and, if necessary, to halt the optimization process using argument **inform**.

If no monitoring is required, **mon** may be specified as **NULLFN**.

mon is called at the end of every i^{th} step where i is controlled by the optional parameter **DFLS Monitor Frequency** (default value 0, **mon** is never called).

The specification of **mon** is:

```
void mon (Integer nvar, const double x[], Integer *inform,
     const double rinfo[], const double stats[], Nag_Comm *comm)
```

1: **nvar** – Integer *Input*

On entry: n, the number of variables in the problem.

2: **x**[**nvar**] – const double *Input*

On entry: the current best point.

3: **inform** – Integer * *Input/Output*

On entry: a non-negative value.

On exit: may be used to request the solver to stop immediately. Specifically, if **inform** < 0 then the value of **rx** will be discarded and the solver will terminate immediately with **fail**.**code** = NE_USER_STOP otherwise, the solver will proceed normally.

4: **rinfo**[**100**] – const double *Input*

On entry: best objective value computed and various indicators (the values are as described in the main argument **rinfo**).

5: **stats**[**100**] – const double *Input*

On entry: solver statistics at the end of the current iteration (the values are as described in the main argument **stats**).

6: **comm** – Nag_Comm *

Pointer to structure of type Nag_Comm; the following members are relevant to **mon**.

user – double *
iuser – Integer *
p – Pointer

The type Pointer will be `void *`. Before calling **nag_opt_handle_solve_dfls (e04ffc)** you may allocate memory and initialize these pointers with various quantities for use by **mon** when called from **nag_opt_handle_solve_dfls (e04ffc)** (see Section 3.3.1.1 in How to Use the NAG Library and its Documentation).

4: **nvar** – Integer *Input*

On entry: n, the number of variables in the problem. It must be unchanged from the value set during the initialization of the handle by **nag_opt_handle_init (e04rac)**.

Constraint: **nvar** ≥ 2.

5: **x**[**nvar**] – double *Input/Output*

On entry: x_0, the initial estimates of the variables x.

On exit: the final values of the variables x.

6: **nres** – Integer *Input*

On entry: m_r, the number of residuals in the problem. It must be unchanged from the value set during the definition of the objective structure by **nag_opt_handle_set_nlnls (e04rmc)**.

7: **rx**[**nres**] – double *Output*

On exit: the values of the residuals at the final point given in **x**.

8: **rinfo**[**100**] – double *Output*

On exit: optimal objective value and various indicators at the end of the final iteration as given in the table below:

0 objective function value $f(x)$ (sum of the squared residuals);

1 ρ, the size of trust region at the end of the algorithm;

2 the number of interpolation points used by the solver.

4 – 101 reserved for future use.

9: **stats**[**100**] – double *Output*

On exit: solver statistics at the end of the final iteration as given in the table below:

0 number of calls to the objective function;

1 if **Stats Time** is activated, total time spent in the solver (including time spent evaluating the objective);

2 if **Stats Time** is activated, total time spent evaluating the objective function;

3 number of steps.

5 – 101 reserved for future use.

10: **comm** – Nag_Comm *

The NAG communication argument (see Section 3.3.1.1 in How to Use the NAG Library and its Documentation).

11: **fail** – NagError * *Input/Output*

The NAG error argument (see Section 3.7 in How to Use the NAG Library and its Documentation).

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.
See Section 2.3.1.2 in How to Use the NAG Library and its Documentation for further information.

NE_BAD_PARAM

On entry, argument ⟨*value*⟩ had an illegal value.

NE_BOUND

Optional argument **DFLS Starting Trust Region** $\rho_{\text{beg}} = \langle value \rangle$, $l_x(i) = \langle value \rangle$, $u_x(i) = \langle value \rangle$ and $i = \langle value \rangle$.
Constraint: if $l_x(i) \neq u_x(i)$ in coordinate i, then $u_x(i) - l_x(i) \geq 2 \times \rho_{\text{beg}}$.
Use **nag_opt_handle_opt_set (e04zmc)** to set compatible option values.

NE_HANDLE

The supplied **handle** does not define a valid handle to the data structure for the NAG optimization modelling suite. It has not been initialized by **nag_opt_handle_init (e04rac)** or it has been corrupted.

NE_INT

There were $n_r = \langle value \rangle$ unequal bounds.
Constraint: $n_r \geq 2$.

There were $n_r = \langle value \rangle$ unequal bounds and the optional argument **DFLS Number Interp Points** $npt = \langle value \rangle$
Constraint: $n_r + 2 \leq npt \leq \frac{(n_r+1)\times(n_r+2)}{2}$.
Use **nag_opt_handle_opt_set (e04zmc)** to set compatible option values.

NE_INTERNAL_ERROR

An internal error has occurred in this function. Check the function call and any array sizes. If the call is correct then please contact NAG for assistance.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

NE_NO_IMPROVEMENT

No progress, the solver was stopped after ⟨*value*⟩ consecutive slow steps.
Use the optional argument **DFLS Maximum Slow Steps** to modify the maximum number of slow steps accepted.

The solver stopped after 5 × **DFLS Maximum Slow Steps** *consecutive slow steps and a trust region above the tolerance set by* **DFLS Trust Region Slow Tol**.

NE_NO_LICENCE

Your licence key may have expired or may not have been installed correctly.
See Section 2.7.5 in How to Use the NAG Library and its Documentation for further information.

NE_PHASE

The problem is already being solved.

NE_REAL_2

Inconsistent optional arguments **DFLS Trust Region Tolerance** ρ_{end} and **DFLS Trust Region Slow Tol** ρ_{tol}.
Constraint: $\rho_{\text{end}} < \rho_{\text{tol}}$.
Use **nag_opt_handle_opt_set (e04zmc)** to set compatible option values.

Inconsistent optional arguments **DFLS Trust Region Tolerance** ρ_{end} and **DFLS Starting Trust Region** ρ_{beg}.
Constraint: $\rho_{\text{end}} < \rho_{\text{beg}}$.
Use **nag_opt_handle_opt_set (e04zmc)** to set compatible option values.

NE_REF_MATCH

The information supplied does not match with that previously stored.
On entry, **nres** $= \langle value \rangle$ must match that given during the definition of the objective in the **handle**, i.e., $\langle value \rangle$.

The information supplied does not match with that previously stored.
On entry, **nvar** $= \langle value \rangle$ must match that given during initialization of the **handle**, i.e., $\langle value \rangle$.

NE_RESCUE_FAILED

A rescue procedure has been called in order to correct damage from rounding errors when computing an update to a quadratic approximation of F, but no further progress could be made. Check your specification of **objfun** and whether the function needs rescaling. Try a different initial **x**.

NE_SETUP_ERROR

The solver does not support the model defined in the handle.
It supports only nonlinear least squares problems with bound constraints.

NE_TIME_LIMIT

The solver terminated after the maximum time allowed was exceeded.

Maximum number of seconds exceeded. Use option **Time Limit** *to reset the limit.*

NE_TOO_MANY_ITER

Maximum number of function evaluations exceeded.

NE_TR_STEP_FAILED

The predicted reduction in a trust region step was non-positive. Check your specification of **objfun** and whether the function needs rescaling. Try a different initial **x**.

NE_USER_STOP

User requested termination after a call to the objective function. **inform** was set to a negative value within the user-supplied function **objfun**.

User requested termination during a monitoring step. **inform** was set to a negative value within the user-supplied function **mon**

NW_NOT_CONVERGED

The problem was solved to an acceptable level after ⟨*value*⟩ consecutive slow iterations. Use the optional argument **DFLS Maximum Slow Steps** to modify the maximum number of slow steps accepted.

The solver stopped after **DFLS Maximum Slow Steps** *consecutive slow steps and a trust region below the tolerance set by* **DFLS Trust Region Slow Tol**.

7 Accuracy

The solver can declare convergence on two conditions:

(i) The trust region radius is below the tolerance ρ_{end} set by the optional parameter **DFLS Trust Region Tolerance**. When this condition is met, the corresponding solution will generally be at a distance lower than $10 \times \rho_{\text{end}}$ of a local minimimum.

(ii) The sum of the square of the residuals is below the tolerance set by the optional parameter **DFLS Small Residuals Tol**. In a data fitting context, this condition means that the error between the observed data and the model is smaller than the requested tolerance.

8 Parallelism and Performance

nag_opt_handle_solve_dfls (e04ffc) makes calls to BLAS and/or LAPACK routines, which may be threaded within the vendor library used by this implementation. Consult the documentation for the vendor library for further information.

Please consult the x06 Chapter Introduction for information on how to control and interrogate the OpenMP environment used within this function. Please also consult the Users' Note for your implementation for any additional implementation-specific information.

9 Further Comments

9.1 Description of the Printed Output

The solver can print information to give an overview of the problem and of the progress of the computation. The output may be sent to two independent file ID which are set by optional parameters **Print File** and **Monitoring File**. Optional parameters **Print Level**, **Print Options**, **Monitoring Level** and **Print Solution** determine the exposed level of detail. This allows, for example, a detailed log file to be generated while the condensed information is displayed on the screen.

By default (**Print File** = 6, **Print Level** = 2), four sections are printed to the standard output: a header, a list of options, an iteration log and a summary.

Header

The header contains statistics about the problem. It should look like:

```
------------------------------------------------
E04FF, Derivative free solver for data fitting
       (nonlinear least squares problems)
------------------------------------------------
Problem statistics
  Number of variables                10
  Number of unconstrained variables  10
  Number of fixed variables           0
  Number of residuals                10
```

Optional parameters list

If **Print Options** is set to YES, a list of the optional parameters and their values is printed. The list shows all options of the solver, each displayed on one line. The line contains the option name, its current value and an indicator for how it was set. The options left at their defaults are noted by (d) and the ones set by the user are noted by (U). Note that the output format is compatible with the file format expected by **nag_opt_handle_opt_set_file (e04zpc)**. The output looks as follows:

```
Stats Time                    =                 Yes     * U
Dfls Trust Region Tolerance   =         1.00000E-07     * U
Dfls Max Objective Calls      =                 500     * d
Dfls Starting Trust Region    =         1.10000E-01     * U
Dfls Number Interp Points     =                   0     * d
```

Iteration log

If **Print Level** ≥ 2, the solver will print a summary line for each step. An iteration is considered successful when it yields a decrease of the objective sufficiently close to the decrease predicted by the quadratic model. The line shows the step number, the value of the objective function, the radius of the trust region and the cumulative number of objective function evaluations. The output looks as follow:

```
---------------------------------------
 step |    obj       rho     |    nf   |
---------------------------------------
    1 |  3.82E+01  1.10E-01  |    13   |
    2 |  3.55E+01  1.10E-01  |    14   |
    3 |  3.05E+01  1.10E-01  |    15   |
    4 |  2.15E+01  1.10E-01  |    18   |
```

Occasionally, the letter 's' is printed at the end of the line indicating that the progress is considered slow by the slow convergence detection heuristic. After a certain number of consecutive slow steps, the solver is stopped. The limit on the number of slow iterations can be controlled by the optional parameter **DFLS Maximum Slow Steps** and the tolerance on the trust region radius before the solver can be stopped is driven by **DFLS Trust Region Slow Tol**.

Summary

Once the solver finishes, a summary is produced:

```
Status: Converged, small trust region size.

Value of the objective                     2.23746E-06
Number of objective function evaluations           107
Number of steps                                     51
```

Optionally, if **Stats Time** is set to YES, the timings are printed:

```
Timings
   Total time spent in the solver            0.056
   Time spent in the objective evaluation    0.012
```

Additionally, if **Print Solution** is set to YES, the solution is printed along with the bounds:

```
Computed Solution:
  idx   Lower bound        Value      Upper bound
    1       -inf       -1.00000E+00        inf
    2       -inf       -1.00000E+00        inf
    3       -inf       -1.00000E+00        inf
    4       -inf       -1.00000E+00        inf
```

10 Example

In this example, we minimize the Kowalik and Osborne function with bounds on some of the variables. In this problem, the number of variables $n = 4$ and the number of residuals $m_r = 11$. The residuals r_i are computed by

$$r_i(x) = z_i - \frac{y_i(y_i + x_2)}{y_i(y_i + x_3) + x_4} x_1 \tag{2}$$

where

$$\begin{aligned} y &= (4.0000, 2.0000, 1.0000, 0.5000, 0.2500, 0.1670, 0.1250, 0.1000, 0.0833, 0.0714, 0.0625) \\ z &= (0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342, 0.0323, 0.0235, 0.0246) \end{aligned} \tag{3}$$

The following bounds are defined on the variables

$$\begin{aligned} 0.2 &\leq x_2 \leq 1.0 \\ 0.3 &\leq x_4 \end{aligned} \tag{4}$$

The initial guess is

$$x_0 = (0.25, 0.39, 0.415, 0.39) \tag{5}$$

The expected solution is

$$x^* = (0.1813, 0.5901, 0.2569, 0.3000) \tag{6}$$

10.1 Program Text

```
/* nag_opt_handle_solve_dfls (e04ffc) Example Program.
 *
 * Copyright 2017 Numerical Algorithms Group.
 *
 * Mark 26.1, 2017.
 */

#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>
#include <nagx04.h>
#include <assert.h>

typedef struct pdata
{
  int    ny, nz;
  double *y, *z;
} pdata;

static void free_pdata(pdata pd);

#ifdef __cplusplus
extern "C"
{
#endif
static void NAG_CALL objfun(Integer nvar, const double x[],
                            Integer nres, double rx[],
                            Integer *inform, Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  const int    defbounds = 1;
  const double infbnd = 1.0e20;

  pdata        pd;
  int          nvar, nres, isparse, nnzrd;
  double       x[4] = { 0.25, 0.39, 0.415, 0.39 };
  double       rinfo[100], stats[100];
  double       *rx, *lx, *ux;
  void         *handle;
  int exit_status = 0;

  /* Nag Types */
  Nag_Comm     comm;
  NagError     fail;

  printf("nag_opt_handle_solve_dfls (e04ffc) Example Program Results\n\n");
  fflush(stdout);

  /* Fill the problem data structure */
  nvar = 4;
  nres = 11;
  pd.ny = nres;
  pd.nz = nres;
  pd.y = NAG_ALLOC(pd.ny,double); assert(pd.y);
  pd.z = NAG_ALLOC(pd.nz,double); assert(pd.z);
```

```
  pd.y[0]  = 4.0   ; pd.z[0] = 0.1957;
  pd.y[1]  = 2.0   ; pd.z[1] = 0.1947;
  pd.y[2]  = 1.0   ; pd.z[2] = 0.1735;
  pd.y[3]  = 0.5   ; pd.z[3] = 0.16;
  pd.y[4]  = 0.25  ; pd.z[4] = 0.0844;
  pd.y[5]  = 0.167 ; pd.z[5] = 0.0627;
  pd.y[6]  = 0.125 ; pd.z[6] = 0.0456;
  pd.y[7]  = 0.1   ; pd.z[7] = 0.0342;
  pd.y[8]  = 0.0833; pd.z[8] = 0.0323;
  pd.y[9]  = 0.0714; pd.z[9] = 0.0235;
  pd.y[10] = 0.0625; pd.z[10] = 0.0246;

  /* nag_opt_handle_init (e04rac).
   * Initialize the handle
   */
  nag_opt_handle_init(&handle, nvar, NAGERR_DEFAULT);

  /* nag_opt_handle_set_nlnls (e04rmc)
   * Define residuals structure, isparse=0 means the residual structure is
   * dense => irowrd and icolrd arguments can be NULL
   */
  isparse = 0;
  nnzrd = 1;
  nag_opt_handle_set_nlnls(handle, nres, isparse, nnzrd, NULL, NULL,
                           NAGERR_DEFAULT);

  /* nag_opt_handle_opt_set (e04zmc)
   * Set options
   */
  /* Relax the main convergence criteria a bit */
  nag_opt_handle_opt_set(handle, "DFLS Trust Region Tolerance = 5.0e-6",
                         NAGERR_DEFAULT);
  /* Turn off option printing */
  nag_opt_handle_opt_set(handle, "Print Options = NO", NAGERR_DEFAULT);
  /* Print the solution */
  nag_opt_handle_opt_set(handle, "Print Solution = X", NAGERR_DEFAULT);

  /* Optionally define bounds for the second and the fourth variable */
  if (defbounds)
    {
      lx = NAG_ALLOC(nvar, double); assert(lx);
      ux = NAG_ALLOC(nvar, double); assert(ux);
      lx[0] = -infbnd; ux[0] = infbnd;
      lx[1] = 0.2; ux[1] = 1.0;
      lx[2] = -infbnd; ux[2] = infbnd;
      lx[3] = 0.3; ux[3] = infbnd;
      /* nag_opt_handle_set_simplebounds (e04rhc) */
      nag_opt_handle_set_simplebounds(handle, nvar, lx, ux, NAGERR_DEFAULT);
    }

  /* nag_opt_handle_solve_dfls (e04ffc)
   * Call the solver
   */
  rx = NAG_ALLOC(nres, double); assert(rx);
  comm.p = &pd;
  INIT_FAIL(fail);
  nag_opt_handle_solve_dfls(handle, objfun, NULL, nvar, x, nres, rx,
                                  rinfo, stats, &comm, &fail);
  if (fail.code != NE_NOERROR){
    printf("Error from nag_opt_handle_solve_dfls (e04ffc).\n%s\n",
           fail.message);
    exit_status = 1;
  }

  /* Clean data */
  if (handle)
    /* nag_opt_handle_free (e04rzc).
     * Destroy the problem handle and deallocate all the memory used
     */
    nag_opt_handle_free(&handle, NAGERR_DEFAULT);
  free_pdata(pd);
```

```
  if (rx)
    NAG_FREE(rx);
  if (lx)
    NAG_FREE(lx);
  if (ux)
    NAG_FREE(ux);

  return exit_status;
}

static void NAG_CALL objfun(Integer nvar, const double x[],
                            Integer nres, double rx[],
                            Integer *inform, Nag_Comm *comm)
{
  pdata  *pd;
  int    i;
  double r1, r2;

  /* Interrupt the solver if the comm structure is not correctly initialized */
  if (!comm || !(comm->p))
    {
      *inform = -1;
      return;
    }

  /* Extract the problem data from the comm structure */
  pd = (pdata *) comm->p;

  /* Interrupt the solver if the data does not correspond to the problem */
  if (nvar != 4 || nres != 11 || pd->ny != nres || pd->nz != nres)
    {
      *inform = -1;
      return;
    }

  /* Fill the residuals values */
  for (i = 0; i < nres; i++)
    {
      r1 = pd->y[i] * (pd->y[i]+x[1]);
      r2 = pd->y[i] * (pd->y[i]+x[2]) + x[3];
      rx[i] = pd->z[i] - x[0]*r1/r2;
    }
}

static void free_pdata(pdata pd)
{
  if (pd.y)
    NAG_FREE(pd.y);
  if (pd.z)
    NAG_FREE(pd.z);
}
```

10.2 Program Data

None.

10.3 Program Results

```
nag_opt_handle_solve_dfls (e04ffc) Example Program Results

  ------------------------------------------
  E04FF, Derivative free solver for data fitting
         (nonlinear least-squares problems)
  ------------------------------------------

  Problem statistics
    Number of variables                 4
    Number of unconstrained variables   2
    Number of fixed variables           0
    Number of residuals                11
```

```
---------------------------------------
 step |    obj       rho     |    nf   |
---------------------------------------
    1 |  1.89E-03  1.00E-01  |     7   |
    2 |  5.77E-04  1.00E-01  |     8   |
    3 |  4.23E-04  1.00E-01  |     9   |
    4 |  4.05E-04  1.00E-02  |    10   |
    5 |  4.02E-04  1.00E-02  |    11   |
    6 |  4.02E-04  1.00E-03  |    16   |
    7 |  4.02E-04  1.00E-03  |    18   |
    8 |  4.02E-04  7.07E-05  |    21   |
    9 |  4.02E-04  7.07E-05  |    22   |
   10 |  4.02E-04  5.00E-06  |    30   |
---------------------------------------
Status: Converged, small trust region size.

Value of the objective                     4.02423E-04
Number of objective function evaluations            30
Number of steps                                     10

Computed Solution:
  idx   Lower bound        Value      Upper bound
    1       -inf       1.81300E-01         inf
    2   2.00000E-01    5.90128E-01    1.00000E+00
    3       -inf       2.56927E-01         inf
    4   3.00000E-01    3.00000E-01         inf
```

11 Algorithmic Details

This section contains a short description of the algorithm used in **nag_opt_handle_solve_dfls (e04ffc)** which is based on Powell's method BOBYQA Powell (2009) and the work of Zhang *et al.* (2010). It is based on a model-based derivative free trust region framework adapted to exploit least squares problems structure.

11.1 Derivative free trust region algorithm

In this section, we are interested in generic problems of the form

$$\underset{x \in R^n}{\text{minimize}} \quad f(x) \tag{7}$$

where the derivatives of the objective function f are not easily available. A model-based derivative free optimization (DFO) algorithm maintains a set of points Y_k centred on an iterate x_k to build quadratic interpolation models of the objective

$$f(x_k + s) \approx \phi_k(s) = f(x_k) + {g_k}^T s + \frac{1}{2} s^T H_k s \tag{8}$$

where g_k and H_k are built with the interpolation conditions

$$y \in Y_k,\ \phi_k(y - x_k) = f(y) \tag{9}$$

Note that if the number of interpolation points npt is smaller than $\frac{(n_r+1)\times(n_r+2)}{2}$, the model chosen is the one for which the hessian H_k is the closest to H_{k-1} in the Frobenius norm sense. This model is iteratively optimized over a trust region, updated and moved around the new computed points. More precisely, it can be described as:

DFO Algorithm

1. **Initialization**

   Choose an initial interpolation set Y_0, trust region radius ρ_{beg} and build the first quadratic model ϕ_0.

2. **Iteration k**

(i) Minimize the model in the trust region to obtain a step s_k.

(ii) If the step is too small, adjust the geometry of the interpolation set and the trust region size ρ_k and restart the iteration.

(iii) Evaluate the objective at the new point $x_k + s_k$.

(iv) Replace a far away point from Y_k by $x_k + s_k$ to create Y_{k+1}.

(v) If the decrease of the objective is sufficient (successful step), choose $x_{k+1} = x_k + s_k$, else choose $x_{k+1} = x_k$.

(vi) Choose ρ_{k+1} and adjust the geometry of Y_{k+1}, if necessary.

(vii) Build ϕ_{k+1} using the new interpolation set.

(viii) Stop the algorithm if $\rho_{k+1}$is below the chosen tolerance ρ_{end}.

In the rest of this documentation page, we call an iteration successful when the trial point $x_k + s_k$ is accepted as the next iterate.

11.2 Bounds on the variables

The bounds on the variables are handled during the model optimization step (step 2(i) of DFO Algorithm) with an active set method. If a bound is hit, it is fixed and step 2(i) is restarted. The set of active constraints is kept throughout the optimization, progressively fixing the corresponding variables.

11.3 Adaptation to nonlinear least squares problems

In the specific case where f is a sum of square $f(x) = \sum_{i=1}^{m_r} r_i(x)^2$, a good approximation of the hessian of the objective can be

$$\nabla^2 f(x) \approx J(x)^T J(x) \tag{10}$$

where J is the m_r by n first derivative matrix of f. This approximation is the main idea behind the Gauss–Newton and Levenberg–Marquardt methods. Following the work of Zhang *et al.* (2010), it is possible to adapt it to the DFO framework. In **nag_opt_handle_solve_dfls (e04ffc)**, one quadratic model is built for each residual r_i

$$r_i(x + s) \approx r_i(x) + {g^{(\mathrm{i})}}^T s + \frac{1}{2} s^T H^{(\mathrm{i})} s \tag{11}$$

We call $J = \left(g^{(1)}, g^{(2)}, ...\right)^T$. To build the model of the objective f, we then choose

$$f(x + s) \approx \phi(s) = f(x) + {g_f}^T s + \frac{1}{2} s^T H_f s \tag{12}$$

where g_f is chosen as

$$g_f = J^T f(x) \tag{13}$$

and H_f as

$$H_f = J^T J + \begin{cases} 0 & \text{if} \quad \left\|g_f\right\| \geq \kappa_1 \\ \kappa_3 \|f(x)\| I & \text{if} \quad \left\|g_f\right\| < \kappa_1 \quad \text{and} \quad \frac{1}{2} f(x) < \kappa_2 \left\|g_f\right\| \\ \sum_{i=1}^{m_r} r_i(x) H^{(\mathrm{i})} & \text{otherwise} \end{cases} \tag{14}$$

The constants κ_1, κ_2 and κ_3 are chosen as proposed in Zhang *et al.* (2010). The first expression amounts to making a Gauss–Newton approximation when we are far from a stationary point, the second to a Levenberg–Marquardt approximation when we are close to a stationary point with small residuals while the third takes the full hessian into account.

nag_opt_handle_solve_dfls (e04ffc) integrates this method of building models into the framework presented in the algorithm DFO Algorithm.

12 Optional Parameters

Several optional parameters in **nag_opt_handle_solve_dfls (e04ffc)** define choices in the problem specification or the algorithm logic. In order to reduce the number of formal arguments of **nag_opt_handle_solve_dfls (e04ffc)** 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 optional parameters can be changed by calling **nag_opt_handle_opt_set (e04zmc)** anytime between the initialization of the handle by **nag_opt_handle_init (e04rac)** and the call to the solver. Modification of the arguments during intermediate monitoring stops is not allowed. Once the solver finishes, the optional parameters can be altered again for the next solve.

The option values may be retrieved by **nag_opt_handle_opt_get (e04znc)**.

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

Defaults
DFLS Maximum Slow Steps
DFLS Max Objective Calls
DFLS Monitor Frequency
DFLS Number Interp Points
DFLS Print Frequency
DFLS Small Residuals Tol
DFLS Starting Trust Region
DFLS Trust Region Slow Tol
DFLS Trust Region Tolerance
DFLS Trust Region Update
Infinite Bound Size
Monitoring File
Monitoring Level
Print File
Print Level
Print Options
Print Solution
Stats Time
Time Limit

12.1 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;

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 (X02AJC)**).

All options accept the value DEFAULT to return single options to their default states.

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. Any argument value given with this keyword will be ignored.

DFLS Maximum Slow Steps i Default $= 20$

If **DFLS Maximum Slow Steps** > 0, this argument defines the maximum number of consecutive slow iterations n_{slow} allowed. Set it to 0 to deactivate the slow iteration detection. The algorithm can stop in two situations:

$n_{\text{slow}} >$ **DFLS Maximum Slow Steps** and $\rho <$ **DFLS Trust Region Slow Tol** with **fail.code** $=$ NW_NOT_CONVERGED

$n_{\text{slow}} > 5 \times$ **DFLS Maximum Slow Steps** with **fail.code** $=$ NE_NO_IMPROVEMENT

Constraint: **DFLS Maximum Slow Steps** ≥ 0.

DFLS Max Objective Calls i Default $= 500$

A limit on the number of objective function evaluations the solver is allowed to compute. If the limit is reached, the solver stops with **fail.code** $=$ NE_TOO_MANY_ITER.

Constraint: **DFLS Max Objective Calls** ≥ 1.

DFLS Monitor Frequency i Default $= 0$

If **DFLS Monitor Frequency** > 0, the solver calls the user defined monitoring function **mon** at the end of every ith step.

Constraint: **DFLS Monitor Frequency** ≥ 0.

DFLS Number Interp Points i Default $= 0$

The number of interpolation points in Y_k (9) used to build the quadratic models. If **DFLS Number Interp Points** $= 0$, the number of points is chosen to be $n_r + 2$ where n_r is the number of non-fixed variables.

Constraint: **DFLS Number Interp Points** ≥ 0.

Consistency constraint, the solver stops with **fail.code** $=$ NE_INT if not met:

$n_r + 2 \leq$ **DFLS Number Interp Points** $\leq \frac{(n_r+1)\times(n_r+2)}{2}$.

DFLS Print Frequency i Default $= 1$

If **DFLS Print Frequency** > 0, the solver prints the iteration log to the appropriate units at the end of every ith step.

Constraint: **DFLS Print Frequency** ≥ 0.

DFLS Small Residuals Tol r Default $= \epsilon^{0.75}$

This option defines the tolerance on the value of the residuals. Namely, the solver declares convergence if

$$f(x) = \sum_{i=1}^{m_r} r_i(x)^2 < \textbf{DFLS Small Residuals Tol}.$$

Constraint: **DFLS Small Residuals Tol** $> \epsilon^2$.

DFLS Starting Trust Region r Default $= 0.1$

ρ_{beg}, the initial trust region radius. This argument should be set to about one tenth of the greatest expected overall change to a variable: the initial quadratic model will be constructed by taking steps from the initial x of length ρ_{beg} along each coordinate direction. The default value assumes that the variables have an order of magnitude 1.

Constraint: **DFLS Starting Trust Region** $> \epsilon$.

Consistency constraints, the solver stops with **fail.code** $=$ NE_BOUND or NE_REAL_2 if not met:

DFLS Starting Trust Region $\leq$ **DFLS Trust Region Tolerance**.

DFLS Starting Trust Region $\leq \frac{1}{2}\min_i(u_x(i) - l_x(i))$

DFLS Trust Region Tolerance r Default $= \epsilon^{0.37}$

ρ_{end}, the requested trust region radius. The algorithm declares convergence when the trust region radius reaches this limit. It should indicate the absolute accuracy that is required in the final values of the variables.

Constraint: **DFLS Trust Region Tolerance** $> \epsilon$.

Consistency constraints, the solver stops with **fail.code** $=$ NE_BOUND or NE_REAL_2 if not met:

DFLS Starting Trust Region $>$ **DFLS Trust Region Tolerance**.

DFLS Trust Region Slow Tol r Default $= \epsilon^{0.25}$

The minimal acceptable trust region radius for the solution to be declared as acceptable. The solver stops if:

$n_{\text{slow}} >$ **DFLS Maximum Slow Steps** and $\rho_k <$ **DFLS Trust Region Slow Tol**

Constraint: **DFLS Trust Region Slow Tol** $> \epsilon$.

Consistency constraints, the solver stops with **fail.code** $=$ NE_BOUND or NE_REAL_2 if not met:

DFLS Trust Region Slow Tol $>$ **DFLS Trust Region Tolerance**

DFLS Trust Region Update a Default $=$ FAST

Controls the speed at which the trust region is decreased after unsuccessful iterations. In smooth non-noisy cases, a fast decrease often leads to faster convergence. However, in noisy cases, a slow decrease is recommended to avoid premature stops.

Constraint: **DFLS Trust Region Update** $=$ FAST or SLOW.

Infinite Bound Size r Default $= 10^{20}$

This defines the 'infinite' bound $bigbnd$ in the definition of the problem constraints. Any upper bound greater than or equal to $bigbnd$ will be regarded as $+\infty$ (and similarly any lower bound less than or equal to $-bigbnd$ will be regarded as $-\infty$). Note that a modification of this optional parameter does not influence constraints which have already been defined; only the constraints formulated after the change will be affected.

Constraint: **Infinite Bound Size** ≥ 1000.

Monitoring File i Default $= -1$

(See Section 3.3.1.1 in How to Use the NAG Library and its Documentation for further information on NAG data types.)

If $i \geq 0$, the Nag_FileID number (as returned from **nag_open_file (x04acc)**) for the secondary (monitoring) output. If set to -1, no secondary output is provided. The information output to this file ID is controlled by **Monitoring Level**.

Constraint: **Monitoring File** ≥ -1.

Monitoring Level i Default $= 4$

This argument sets the amount of information detail that will be printed by the solver to the secondary output. The meaning of the levels is the same as with **Print Level**.

Constraint: $0 \leq$ **Monitoring Level** ≤ 5.

Print File i Default
$=$ Nag_FileID number associated with stdout

(See Section 3.3.1.1 in How to Use the NAG Library and its Documentation for further information on NAG data types.)

If $i \geq 0$, the Nag_FileID number (as returned from **nag_open_file (x04acc)**, `stdout` as the default) for the primary output of the solver. If **Print File** $= -1$, the primary output is completely turned off independently of other settings. The information output to this unit is controlled by **Print Level**.

Constraint: **Print File** ≥ -1.

Print Level i Default $= 2$

This argument defines how detailed information should be printed by the solver to the primary and secondary output.

i	**Output**
0	No output from the solver
1	The Header and Summary.
2,3,4,5	Additionally, the Iteration log.

Constraint: $0 \leq$ **Print Level** ≤ 5.

Print Options a Default $=$ YES

If **Print Options** $=$ YES, a listing of optional parameters will be printed to the primary output. It is always printed to the secondary output.

Constraint: **Print Options** $=$ YES or NO.

Print Solution a Default $=$ NO

If **Print Solution** $=$ YES, the solution will be printed to the primary and secondary output.

Constraint: **Print Solution** $=$ NO or YES.

Stats Time a Default $=$ NO

This argument turns on timings of various parts of the algorithm to give a better overview of where most of the time is spent. This might be helpful for a choice of different solving approaches. It is possible to choose between CPU and wall clock time. Choice YES is equivalent to wall clock.

Constraint: **Stats Time** $=$ YES, NO, CPU or WALL CLOCK.

Time Limit r Default $= 10^6$

A limit on seconds that the solver can use to solve one problem. If during the convergence check this limit is exceeded, the solver will terminate with **fail.code** $=$ NE_TIME_LIMIT error message.

Warning: the timings are not computed if **Stats Time** is set to NO. The solver will therefore NOT be stopped if the time limit is exceeded in such a case.

Constraint: **Time Limit** > 0.
