NAG Library Function Document

nag_opt_nlp_revcomm_option_set_file (e04udc)

1 Purpose

To supply optional arguments to nag_opt_nlp_revcomm (e04ufc) from an external file.

2 Specification

```
#include <nag.h>
#include <nage04.h>
```

```
void nag_opt_nlp_revcomm_option_set_file (Nag_FileID fileid,
     Nag_Boolean lwsav[], Integer iwsav[], double rwsav[], NagError *fail)
```

3 Description

nag_opt_nlp_revcomm_option_set_file (e04udc) may be used to supply values for optional arguments to nag_opt_nlp_revcomm (e04ufc). nag_opt_nlp_revcomm_option_set_file (e04udc) reads an external file and each line of the file defines a single optional argument. It is only necessary to supply values for those arguments whose values are to be different from their default values.

Each optional argument is defined by a single character string, consisting of one or more items. The items associated with a given option must be separated by spaces, or equals signs [=]. Alphabetic characters may be upper or lower case. The string

```
Print Level = 1
```

is an example of a string used to set an optional argument. For each option the string contains one or more of the following items:

- a mandatory keyword;
- a phrase that qualifies the keyword;
- a number that specifies an Integer or double value. Such numbers may be up to 16 contiguous characters which can be read using C's `d` or `g` formats, terminated by a space if this is not the last item on the line.

Blank strings and comments are ignored. A comment begins with an asterisk (*) and all subsequent characters in the string are regarded as part of the comment.

The file containing the options must start with `Begin` and must finish with `End`. An example of a valid options file is:

```
Begin * Example options file
   Print level = 5
End
```

Optional argument settings are preserved following a call to nag_opt_nlp_revcomm (e04ufc) and so the keyword **Defaults** is provided to allow you to reset all the optional arguments to their default values before a subsequent call to nag_opt_nlp_revcomm (e04ufc).

A complete list of optional arguments, their abbreviations, synonyms and default values is given in Section 12 in nag_opt_nlp_revcomm (e04ufc).

4 References

None.

5 Arguments

1: **fileid** – Nag_FileID *Input*

On entry: the ID of the option file to be read as returned by a call to nag_open_file (x04acc).

2: **lwsav**[**120**] – Nag_Boolean *Communication Array*
3: **iwsav**[**610**] – Integer *Communication Array*
4: **rwsav**[**475**] – double *Communication Array*

The arrays **lwsav**, **iwsav** and **rwsav** MUST NOT be altered between calls to any of the functions nag_opt_nlp_revcomm_option_set_file (e04udc), nag_opt_nlp_revcomm_option_set_string (e04uec), nag_opt_nlp_revcomm (e04ufc) or nag_opt_nlp_revcomm_init (e04wbc).

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

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

6 Error Indicators and Warnings

NE_ALLOC_FAIL

Dynamic memory allocation failed.

NE_BAD_PARAM

On entry, argument $\langle value\rangle$ had an illegal value.

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.

NE_INVALID_OPTION

One or more lines of the options file is invalid.

NE_MISSING_BEGIN

End-of-file was found before `Begin` was found.

NE_MISSING_END

`Begin` was found, but end-of-file was found before `End` was found.

NE_NOT_READ_FILE

Could not open options file with **fileid** $= \langle value\rangle$.

7 Accuracy

Not applicable.

8 Parallelism and Performance

Not applicable.

9 Further Comments

nag_opt_nlp_revcomm_option_set_string (e04uec) may also be used to supply optional arguments to nag_opt_nlp_revcomm (e04ufc).

10 Example

This example solves the same problem as the example for nag_opt_nlp_revcomm (e04ufc), but in addition illustrates the use of nag_opt_nlp_revcomm_option_set_file (e04udc) and nag_opt_nlp_revcomm_option_set_string (e04uec) to set optional arguments for nag_opt_nlp_revcomm (e04ufc).

10.1 Program Text

```
/* nag_opt_nlp_revcomm_option_set_file (e04udc) Example Program.
 *
 * Copyright 2011, Numerical Algorithms Group.
 *
 * Mark 23, 2011
 *
 */
#include <stdio.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>

int main(void)
{
  const char   *optionsfile = "e04udce.opt";

  /* Scalars */
  double        objf, nctotal;
  Integer       exit_status=0, i, irevcm, iter, j, n, nclin, ncnln;
  Integer       tda, tdcj, tdr, licomm, lrcomm;

  /* Arrays */
  double        *a=0, *bl=0, *bu=0, *c=0, *cjac=0, *clamda=0, *objgrd=0;
  double        *r=0, *rcomm=0, rwsav[475], *x=0;
  Integer       *icomm=0, *istate=0, iwsav[610], *needc = 0;
  char          cwsav[5*80];

  /* Nag Types */
  Nag_Boolean   lwsav[120];
  NagError      fail;
  Nag_FileID    optfileid;

#define A(I,J) a[(I-1)*tda + J - 1]
#define CJAC(I,J) cjac[(I-1)*tdcj + J - 1]

  INIT_FAIL(fail);

  printf("nag_opt_nlp_revcomm_option_set_file (e04udc) "
         "Example Program Results\n");
  fflush(stdout);

  /* Skip heading in data file */
  scanf("%*[^\n] ");
  scanf("%ld%ld%ld%*[^\n] ", &n, &nclin, &ncnln);

  if (n <= 0 || nclin < 0 || ncnln < 0)
    {
      printf("At least one of n, nclin, or ncnln is invalid\n");
      exit_status = 1;
    }
  else
    {
      tda = MAX(nclin,n);
      tdcj = MAX(ncnln,n);
      tdr = n;
      nctotal = n + nclin + ncnln;
      licomm = 3*n + nclin + 2*ncnln;
      lrcomm = 21*n + 2;
      if (ncnln || nclin) lrcomm += 2*n*n + 11*nclin;
      if (ncnln) lrcomm += n*nclin + 2*n*ncnln + 22*ncnln - 1;
```

```
/* Allocate memory */
if (!(a = NAG_ALLOC(tda*MAX(1,nclin), double)) ||
    !(bl = NAG_ALLOC(nctotal, double)) ||
    !(bu = NAG_ALLOC(nctotal, double)) ||
    !(istate = NAG_ALLOC(nctotal, Integer)) ||
    !(c = NAG_ALLOC(ncnln, double)) ||
    !(cjac = NAG_ALLOC(tdcj*MAX(1,ncnln), double)) ||
    !(clamda = NAG_ALLOC(nctotal, double)) ||
    !(objgrd = NAG_ALLOC(n, double)) ||
    !(r = NAG_ALLOC(tdr*n, double)) ||
    !(x = NAG_ALLOC(n, double)) ||
    !(needc = NAG_ALLOC(ncnln, Integer)) ||
    !(icomm = NAG_ALLOC(licomm, Integer)) ||
    !(rcomm = NAG_ALLOC(lrcomm, double)))
  {
    printf("Allocation failure\n");
    exit_status = -1;
  }
else
  {
    /* Read A, BL, BU and X from data file */
    if (nclin > 0)
      {
        for (i = 1; i <= nclin; ++i)
          for (j = 1; j <= n; ++j)
            scanf("%lf", &A(i, j));
        scanf("%*[^\n] ");
      }

    for (i = 0; i < nctotal; ++i)
      scanf("%lf", &bl[i]);
    scanf("%*[^\n] ");
    for (i = 0; i < nctotal; ++i)
      scanf("%lf", &bu[i]);
    scanf("%*[^\n] ");
    for (i = 0; i < n; ++i)
      scanf("%lf", &x[i]);

    /* Set all constraint Jacobian elements to zero.
       Note that this will only work when 'Derivative Level = 3'
       (the default; see Section 11.2) */

    for (j = 1; j <= n; ++j)
      for (i = 1; i <= ncnln; ++i)
        CJAC(i,j) = 0.0;

    /* Initialise nag_opt_nlp_revcomm (e04ufc) and check for error
       exits */
    nag_opt_nlp_revcomm_init("e04ufc",cwsav,5,lwsav,120,iwsav,610,
                             rwsav,475,&fail);

    /* Set three options using
       nag_opt_nlp_revcomm_option_set_string (e04uec) */
    nag_opt_nlp_revcomm_option_set_string("Infinite Bound Size = 1.0d+25",
                                          lwsav,iwsav,rwsav,&fail);
    nag_opt_nlp_revcomm_option_set_string("Print Level = 1",lwsav,iwsav,
                                          rwsav,&fail);
    nag_opt_nlp_revcomm_option_set_string("Verify Level = 11",lwsav,iwsav,
                                          rwsav,&fail);

    /* Use nag_opt_sparse_nlp_option_set_file (e04vkc) to read some
     * options from the options file. Call nag_open_file (x04acc) to
     * set the options file optfileid */
    /* nag_open_file (x04acc), see above. */
    nag_open_file(optionsfile, 0, &optfileid, &fail);

    /* nag_opt_nlp_revcomm_option_set_file (e04udc).
     * Supply optional parameter values for
     * nag_opt_nlp_revcomm (e04ufc) from external file
     */
```

```
      nag_opt_nlp_revcomm_option_set_file(optfileid,lwsav,iwsav,rwsav,
                                          &fail);

      /* Solve the problem */
      irevcm = 0;

      do
        {
          nag_opt_nlp_revcomm(&irevcm,n,nclin,ncnln,tda,tdcj,tdr,a,
                              bl,bu,&iter,istate,c,cjac,clamda,&objf,objgrd,
                              r,x,needc,icomm,licomm,rcomm,lrcomm,cwsav,
                              lwsav,iwsav,rwsav,&fail);

          if (irevcm == 1 || irevcm == 3)
            /* Evaluate the objective function */
            objf = x[0]*x[3]*(x[0]+x[1]+x[2]) + x[2];

          if (irevcm == 2 || irevcm == 3)
            {
              /* Evaluate the objective gradient */
              objgrd[0] = x[3]*(2.0*x[0]+x[1]+x[2]);
              objgrd[1] = x[0]*x[3];
              objgrd[2] = x[0]*x[3] + 1.0;
              objgrd[3] = x[0]*(x[0]+x[1]+x[2]);
            }

          if (irevcm == 4 || irevcm == 6)
            {
              /* Evaluate the nonlinear constraint functions */
              if (needc[0] > 0)
                c[0] = x[0]*x[0] + x[1]*x[1] + x[2]*x[2] + x[3]*x[3];
              if (needc[1] > 0)
                c[1] = x[0]*x[1]*x[2]*x[3];
            }

          if (irevcm == 5 || irevcm == 6)
            {
              /* Evaluate the constraint Jacobian */
              if (needc[0] > 0)
                {
                  CJAC(1,1) = 2.0*x[0];
                  CJAC(1,2) = 2.0*x[1];
                  CJAC(1,3) = 2.0*x[2];
                  CJAC(1,4) = 2.0*x[3];
                }

              if (needc[1] > 0)
                {
                  CJAC(2,1) = x[1]*x[2]*x[3];
                  CJAC(2,2) = x[0]*x[2]*x[3];
                  CJAC(2,3) = x[0]*x[1]*x[3];
                  CJAC(2,4) = x[0]*x[1]*x[2];
                }
            }
        }  while (irevcm > 0);

      if (fail.code != NE_NOERROR)
        {
          printf("e04ufc failed.\n%s\n",fail.message);
          exit_status = 1;
        }
    }

  /* Deallocate any allocated arrays */
  NAG_FREE(a);
  NAG_FREE(bl);
  NAG_FREE(bu);
  NAG_FREE(istate);
  NAG_FREE(c);
  NAG_FREE(cjac);
  NAG_FREE(clamda);
```

```
      NAG_FREE(objgrd);
      NAG_FREE(r);
      NAG_FREE(x);
      NAG_FREE(needc);
      NAG_FREE(icomm);
      NAG_FREE(rcomm);
    }
  return exit_status;
}
```

10.2 Program Data

```
nag_opt_nlp_revcomm_option_set_file (e04udc) Example Program Data
  4   1   2                              :Values of n, nclin and ncnln
  1.0   1.0   1.0   1.0                              :End of matrix a
  1.0   1.0   1.0   1.0  -1.0e+25  -1.0e+25  25.0      :End of bl
  5.0   5.0   5.0   5.0  20.0      40.0       1.0e+25   :End of bu
  1.0   5.0   5.0   1.0                              :End of x

Begin *  Example options file for e04ufc
   Major Iteration Limit   =  15      *  (Default =  50)
   Minor Iteration Limit   =  10      *  (Default =  50)
End
```

10.3 Program Results

```
nag_opt_nlp_revcomm_option_set_file (e04udc) Example Program Results

 *** e04ufc

 Parameters
 ----------

 Linear constraints.....         1       Variables..............         4
 Nonlinear constraints..         2

 Infinite bound size....  1.00E+25       COLD start.............
 Infinite step size.....  1.00E+25       EPS (machine precision)  1.11E-16
 Step limit.............  2.00E+00       Hessian................        NO

 Linear feasibility.....  1.05E-08       Crash tolerance........  1.00E-02
 Nonlinear feasibility..  1.05E-08       Optimality tolerance...  3.26E-12
 Line search tolerance..  9.00E-01       Function precision.....  4.37E-15

 Derivative level.......         3       Monitoring file........        -1
 Verify level...........        11

 Start obj chk at varble         1       Stop obj chk at varble.         4
 Start con chk at varble         1       Stop con chk at varble.         4

 Major iterations limit.        15       Major print level......         1
 Minor iterations limit.        10       Minor print level......         0

 Workspace provided is     IWORK(      17),  WORK(     192).
 To solve problem we need  IWORK(      17),  WORK(     192).


 Verification of the objective gradients.
 ----------------------------------------

 The objective gradients seem to be ok.

 Directional derivative of the objective    8.15250000E-01
 Difference approximation                   8.15249734E-01


    J    X(J)     DX(J)           G(J)           Difference approxn  Itns

    1  1.00E+00  3.86E-07    1.20000000E+01    1.20000000E+01    OK     1
    2  5.00E+00  7.94E-06    1.00000000E+00    1.00000000E+00    OK     3
```

```
   3  5.00E+00  7.94E-06    2.00000000E+00    2.00000000E+00    OK     3
   4  1.00E+00  2.65E-06    1.10000000E+01    1.10000000E+01    OK     3

     4  Objective gradients out of the     4
        set in cols     1  through     4  seem to be ok.

The largest relative error was    5.54E-12   in element     1


Exit from NP problem after     5 major iterations,
                               9 minor iterations.


Varbl State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

V   1    LL    1.00000       1.00000       5.00000       1.088         .
V   2    FR    4.74300       1.00000       5.00000           .      0.2570
V   3    FR    3.82115       1.00000       5.00000           .       1.179
V   4    FR    1.37941       1.00000       5.00000           .      0.3794


L Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

L   1    FR    10.9436          None       20.0000           .       9.056


N Con State     Value       Lower Bound   Upper Bound    Lagr Mult   Slack

N   1    UL    40.0000          None       40.0000     -0.1615     -3.5264E-11
N   2    LL    25.0000       25.0000          None      0.5523     -2.8791E-11

Exit e04ufc - Optimal solution found.

Final objective value =    17.01402
```