NAG Library Function Document

nag_opt_sparse_nlp_option_set_file (e04vkc)

1 Purpose

nag_opt_sparse_nlp_option_set_file (e04vkc) may be used to supply optional parameters to nag_opt_sparse_nlp_solve (e04vhc) from an external file. The initialization function nag_opt_sparse_nlp_init (e04vgc) **must** have been called before calling nag_opt_sparse_nlp_option_set_file (e04vkc).

2 Specification

```
#include <nag.h>
#include <nage04.h>
void nag_opt_sparse_nlp_option_set_file (Nag_FileID fileid,
     Nag_E04State *state, NagError *fail)
```

3 Description

nag_opt_sparse_nlp_option_set_file (e04vkc) may be used to supply values for optional parameters to nag_opt_sparse_nlp_solve (e04vhc). nag_opt_sparse_nlp_option_set_file (e04vkc) reads an external file whose **fileid** has been returned by a call to nag_open_file (x04acc). nag_open_file (x04acc) must be called to provide **fileid**. Each line of the file defines a single optional parameter. It is only necessary to supply values for those arguments whose values are to be different from their default values.

Each optional parameter 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 parameter. 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 parameter settings are preserved following a call to nag_opt_sparse_nlp_solve (e04vhc) and so the keyword **Defaults** is provided to allow you to reset all the optional parameters to their default values before a subsequent call to nag_opt_sparse_nlp_solve (e04vhc).

A complete list of optional parameters, their abbreviations, synonyms and default values is given in Section 12 in nag_opt_sparse_nlp_solve (e04vhc).

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: **state** – Nag_E04State * *Communication Structure*

state contains internal information required for functions in this suite. It must not be modified in any way.

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

The NAG error argument (see Section 2.7 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_E04VGC_NOT_INIT

The initialization function nag_opt_sparse_nlp_init (e04vgc) has not been called.

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.

An unexpected error has been triggered by this function. Please contact NAG.
See Section 2.7.6 in How to Use the NAG Library and its Documentation for further information.

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_OPTIONS_FILE_READ_FAILURE

At least one line of the options file is invalid.

Could not read options file on unit **fileid** = ⟨*value*⟩.

Could not read options file on unit **fileid**. *This may be due to:*

(a) **fileid** is not a valid unit number;

(b) a file is not associated with unit **fileid**, or if it is, is unavailable for read access;

(c) one or more lines of the options file is invalid. Check that all keywords are neither ambiguous nor misspelt;

(d) `Begin` was found, but end-of-file was found before `End` was found;

(e) end-of-file was found before `Begin` was found.

7 Accuracy

Not applicable.

8 Parallelism and Performance

nag_opt_sparse_nlp_option_set_file (e04vkc) is not threaded in any implementation.

9 Further Comments

nag_opt_sparse_nlp_option_set_string (e04vlc), nag_opt_sparse_nlp_option_set_integer (e04vmc) or nag_opt_sparse_nlp_option_set_double (e04vnc) may also be used to supply optional parameters to nag_opt_sparse_nlp_solve (e04vhc).

10 Example

This example solves the same problem as the example in the document for nag_opt_sparse_nlp_solve (e04vhc), but sets and reads some optional parameters first. See Section 10 in nag_opt_sparse_nlp_solve (e04vhc) for further details.

The example in the document for nag_opt_sparse_nlp_jacobian (e04vjc) also solves the same problem (see Section 10 in nag_opt_sparse_nlp_jacobian (e04vjc)), but it first calls nag_opt_sparse_nlp_jacobian (e04vjc) to determine the sparsity pattern before calling nag_opt_sparse_nlp_option_set_file (e04vkc).

10.1 Program Text

```
/* nag_opt_sparse_nlp_option_set_file (e04vkc) Example Program.
 *
 * NAGPRODCODE Version.
 *
 * Copyright 2016 Numerical Algorithms Group.
 *
 * Mark 26, 2016.
 */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include <nag.h>
#include <nag_stdlib.h>
#include <nage04.h>

#ifdef __cplusplus
extern "C"
{
#endif
  static void NAG_CALL usrfun(Integer *status, Integer n, const double x[],
                              Integer needf, Integer nf, double f[],
                              Integer needg, Integer leng, double g[],
                              Nag_Comm *comm);
#ifdef __cplusplus
}
#endif

int main(void)
{
  const char *optionsfile = "e04vkce.opt";

  /* Scalars */
  double bndinf, featol, objadd, sinf;
  Integer elmode, exit_status = 0, i, lena, leng, n, nea, neg, nf, nfname,
          ninf;
```

```
  Integer ns, nxname, objrow;

  /* Arrays */
  static double ruser[1] = { -1.0 };
  char nag_enum_arg[40];
  char **fnames = 0, *prob = 0, **xnames = 0;
  double *a = 0, *f = 0, *flow = 0, *fmul = 0, *fupp = 0;
  double *x = 0, *xlow = 0, *xmul = 0, *xupp = 0;
  Integer *fstate = 0, *iafun = 0, *igfun = 0, *iuser = 0, *javar = 0;
  Integer *jgvar = 0, *xstate = 0;

  /* Nag Types */
  Nag_E04State state;
  NagError fail;
  Nag_Comm comm;
  Nag_Start start;
  Nag_FileID optfileid;

  /* By default e04vhc does not print monitoring information.
     Define SHOW_MONITORING_INFO to turn it on - see further below. */
#ifdef SHOW_MONITORING_INFO
  Nag_FileID outfileid;
#endif

  INIT_FAIL(fail);

  printf("%s\n",
         "nag_opt_sparse_nlp_option_set_file (e04vkc) Example Program"
         " Results");

  /* For communication with user-supplied functions: */
  comm.user = ruser;

  fflush(stdout);

  /* This program demonstrates the use of routines to set and get values of
   * optional parameters associated with nag_opt_sparse_nlp_solve (e04vhc).
   */

  /* Skip heading in data file */
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &n, &nf);
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &n, &nf);
#endif
#ifdef _WIN32
  scanf_s("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT " %39s %*[^\n] ", &nea, &neg,
          &objrow, nag_enum_arg, (unsigned)_countof(nag_enum_arg));
#else
  scanf("%" NAG_IFMT "%" NAG_IFMT "%" NAG_IFMT " %39s %*[^\n] ", &nea, &neg,
        &objrow, nag_enum_arg);
#endif

  /* nag_enum_name_to_value (x04nac).
   * Converts NAG enum member name to value
   */
  start = (Nag_Start) nag_enum_name_to_value(nag_enum_arg);

  if (n > 0 && nf > 0 && nea > 0 && neg > 0) {
    nxname = n;
    nfname = nf;

    /* Allocate memory */
    if (!(fnames = NAG_ALLOC(nfname, char *)) ||
        !(prob = NAG_ALLOC(9, char)) ||
        !(xnames = NAG_ALLOC(nxname, char *)) ||
```

```
        !(a = NAG_ALLOC(300, double)) ||
        !(f = NAG_ALLOC(100, double)) ||
        !(flow = NAG_ALLOC(100, double)) ||
        !(fmul = NAG_ALLOC(100, double)) ||
        !(fupp = NAG_ALLOC(100, double)) ||
        !(x = NAG_ALLOC(100, double)) ||
        !(xlow = NAG_ALLOC(100, double)) ||
        !(xmul = NAG_ALLOC(100, double)) ||
        !(xupp = NAG_ALLOC(100, double)) ||
        !(fstate = NAG_ALLOC(100, Integer)) ||
        !(iafun = NAG_ALLOC(300, Integer)) ||
        !(igfun = NAG_ALLOC(300, Integer)) ||
        !(iuser = NAG_ALLOC(1, Integer)) ||
        !(javar = NAG_ALLOC(300, Integer)) ||
        !(jgvar = NAG_ALLOC(300, Integer)) ||
        !(xstate = NAG_ALLOC(100, Integer)))
    {
      printf("Allocation failure\n");
      exit_status = -1;
      goto END;
    }
  }
  else {
    printf("Invalid n or nf or nea or neg\n");
    exit_status = 1;
    return exit_status;
  }
  lena = MAX(1, nea);
  leng = MAX(1, neg);
  objadd = 0.;
#ifdef _WIN32
  strcpy_s(prob, 9, "        ");
#else
  strcpy(prob, "        ");
#endif

  /* Read the variable names xnames */

  for (i = 0; i < nxname; ++i) {
    xnames[i] = NAG_ALLOC(9, char);
#ifdef _WIN32
    scanf_s(" ' %8s '", xnames[i], 9);
#else
    scanf(" ' %8s '", xnames[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read the function names fnames */
  for (i = 0; i < nfname; ++i) {
    fnames[i] = NAG_ALLOC(9, char);
#ifdef _WIN32
    scanf_s(" '%8s'", fnames[i], 9);
#else
    scanf(" '%8s'", fnames[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Read the sparse matrix A, the linear part of F */
  for (i = 0; i < nea; ++i) {
    /* For each element read row, column, A(row,column) */
#ifdef _WIN32
```

```
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%lf%*[^\n] ", &iafun[i], &javar[i],
            &a[i]);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%lf%*[^\n] ", &iafun[i], &javar[i],
          &a[i]);
#endif
  }
  /* Read the structure of sparse matrix g, the nonlinear part of f */
  for (i = 0; i < neg; ++i) {
    /* For each element read row, column */
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &igfun[i], &jgvar[i]);
#else
    scanf("%" NAG_IFMT "%" NAG_IFMT "%*[^\n] ", &igfun[i], &jgvar[i]);
#endif
  }

  /* Read the lower and upper bounds on the variables */
  for (i = 0; i < n; ++i) {
#ifdef _WIN32
    scanf_s("%lf%lf%*[^\n] ", &xlow[i], &xupp[i]);
#else
    scanf("%lf%lf%*[^\n] ", &xlow[i], &xupp[i]);
#endif
  }

  /* Read the lower and upper bounds on the functions */
  for (i = 0; i < nf; ++i) {
#ifdef _WIN32
    scanf_s("%lf%lf%*[^\n] ", &flow[i], &fupp[i]);
#else
    scanf("%lf%lf%*[^\n] ", &flow[i], &fupp[i]);
#endif
  }

  /* Initialize x, xstate, xmul, f, fstate, fmul */
  for (i = 0; i < n; ++i) {
#ifdef _WIN32
    scanf_s("%lf", &x[i]);
#else
    scanf("%lf", &x[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  for (i = 0; i < n; ++i) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &xstate[i]);
#else
    scanf("%" NAG_IFMT "", &xstate[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  for (i = 0; i < n; ++i) {
#ifdef _WIN32
    scanf_s("%lf", &xmul[i]);
#else
    scanf("%lf", &xmul[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
```

```
#else
  scanf("%*[^\n] ");
#endif

  for (i = 0; i < nf; ++i) {
#ifdef _WIN32
    scanf_s("%lf", &f[i]);
#else
    scanf("%lf", &f[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  for (i = 0; i < nf; ++i) {
#ifdef _WIN32
    scanf_s("%" NAG_IFMT "", &fstate[i]);
#else
    scanf("%" NAG_IFMT "", &fstate[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  for (i = 0; i < nf; ++i) {
#ifdef _WIN32
    scanf_s("%lf", &fmul[i]);
#else
    scanf("%lf", &fmul[i]);
#endif
  }
#ifdef _WIN32
  scanf_s("%*[^\n] ");
#else
  scanf("%*[^\n] ");
#endif

  /* Initialize e04vhc using nag_opt_sparse_nlp_init (e04vgc):
   *  Initialization function for nag_opt_sparse_nlp_solve (e04vhc).
   */
  nag_opt_sparse_nlp_init(&state, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Initialization of nag_opt_sparse_nlp_init (e04vgc) failed.\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }

#ifdef SHOW_MONITORING_INFO
  /* Call nag_open_file (x04acc) to set the print file outfileid */
  /* nag_open_file (x04acc).
   * Open unit number for reading, writing or appending, and
   * associate unit with named file
   */
  nag_open_file("", 2, &outfileid, &fail);
  if (fail.code != NE_NOERROR) {
    exit_status = 2;
    goto END;
  }

  /* nag_opt_sparse_nlp_option_set_integer (e04vmc).
   * Set a single option for nag_opt_sparse_nlp_solve (e04vhc)
   * from an integer argument
   */
  nag_opt_sparse_nlp_option_set_integer("Print file", outfileid, &state,
```

```
                                        &fail);

  if (fail.code != NE_NOERROR) {
    exit_status = 1;
    goto END;
  }
#endif

  /* 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(optionsfile, 0, &optfileid, &fail);
  if (fail.code != NE_NOERROR) {
    nag_close_file(optfileid, &fail);
    exit_status = 1;
    goto END;
  }
  /* nag_opt_sparse_nlp_option_set_file (e04vkc).
   * Supply optional parameter values for
   * nag_opt_sparse_nlp_solve (e04vhc) from external file
   */
  nag_opt_sparse_nlp_option_set_file(optfileid, &state, &fail);
  if (fail.code != NE_NOERROR) {
    nag_close_file(optfileid, &fail);
    exit_status = 1;
    goto END;
  }
  printf("\n");

  /* Find the value of Integer-valued option 'Elastic mode' using
   * nag_opt_sparse_nlp_option_get_integer (e04vrc):
   *  Get the setting of an integer valued option of
   *  nag_opt_sparse_nlp_solve (e04vhc)
   */
  nag_opt_sparse_nlp_option_get_integer("Elastic mode", &elmode, &state,
                                        &fail);
  if (fail.code != NE_NOERROR) {
    nag_close_file(optfileid, &fail);
    exit_status = 1;
    goto END;
  }
  printf("Option 'Elastic mode' has the value %3" NAG_IFMT ".\n", elmode);

  /* Use nag_opt_sparse_nlp_option_set_double (e04vnc) to set the value of
   *  real-valued option 'Infinite bound size'.
   */
  bndinf = 1e10;
  /* nag_opt_sparse_nlp_option_set_double (e04vnc).
   * Set a single option for nag_opt_sparse_nlp_solve (e04vhc)
   * from a double argument
   */
  nag_opt_sparse_nlp_option_set_double("Infinite bound size", bndinf, &state,
                                       &fail);
  if (fail.code != NE_NOERROR) {
    nag_close_file(optfileid, &fail);
    exit_status = 1;
    goto END;
  }

  /* Find the value of real-valued option 'Feasibility tolerance' using
   * nag_opt_sparse_nlp_option_get_double (e04vsc):
   *  Get the setting of a double valued option of
   *  nag_opt_sparse_nlp_solve (e04vhc)
   */
  nag_opt_sparse_nlp_option_get_double("Feasibility tolerance", &featol,
                                       &state, &fail);
  if (fail.code != NE_NOERROR) {
    nag_close_file(optfileid, &fail);
    exit_status = 1;
    goto END;
```

```
  }
  printf("Option 'Feasibility tolerance' has the value %14.5e.\n", featol);

  /* Set the option 'Major iterations limit' using
   * nag_opt_sparse_nlp_option_set_string (e04vlc):
   *  Set a single option for nag_opt_sparse_nlp_solve (e04vhc)
   *  from a character string
   */
  nag_opt_sparse_nlp_option_set_string("Major iterations limit 50", &state,
                                       &fail);
  if (fail.code != NE_NOERROR) {
    nag_close_file(optfileid, &fail);
    exit_status = 1;
    goto END;
  }
  printf("\n");
  fflush(stdout);

  /* Solve the problem. */
  /* nag_opt_sparse_nlp_solve (e04vhc).
   * General sparse nonlinear optimizer
   */
  nag_opt_sparse_nlp_solve(start, nf, n, nxname, nfname, objadd, objrow, prob,
                           usrfun, iafun, javar, a, lena, nea, igfun, jgvar,
                           leng, neg, xlow, xupp, (const char **) xnames,
                           flow, fupp, (const char **) fnames, x, xstate,
                           xmul, f, fstate, fmul, &ns, &ninf, &sinf, &state,
                           &comm, &fail);
  if (fail.code != NE_NOERROR) {
    printf("Error from nag_opt_sparse_nlp_solve (e04vhc).\n%s\n",
           fail.message);
    exit_status = 1;
    goto END;
  }
  nag_close_file(optfileid, &fail);

  printf("Final objective value = %11.1f\n", f[objrow - 1]);
  printf("Optimal X = ");

  for (i = 0; i < n; ++i)
    printf("%9.2f%s", x[i], i % 7 == 6 || i == n - 1 ? "\n" : " ");

END:
  for (i = 0; i < nxname; i++)
    NAG_FREE(xnames[i]);
  for (i = 0; i < nfname; i++)
    NAG_FREE(fnames[i]);
  NAG_FREE(fnames);
  NAG_FREE(xnames);
  NAG_FREE(prob);
  NAG_FREE(a);
  NAG_FREE(f);
  NAG_FREE(flow);
  NAG_FREE(fmul);
  NAG_FREE(fupp);
  NAG_FREE(x);
  NAG_FREE(xlow);
  NAG_FREE(xmul);
  NAG_FREE(xupp);
  NAG_FREE(fstate);
  NAG_FREE(iafun);
  NAG_FREE(igfun);
  NAG_FREE(iuser);
  NAG_FREE(javar);
  NAG_FREE(jgvar);
  NAG_FREE(xstate);

  return exit_status;
}

static void NAG_CALL usrfun(Integer *status, Integer n, const double x[],
```

```
                           Integer needf, Integer nf, double f[],
                           Integer needg, Integer leng, double g[],
                           Nag_Comm *comm)
{
  if (comm->user[0] == -1.0) {
    fflush(stdout);
    printf("(User-supplied callback usrfun, first invocation.)\n");
    comm->user[0] = 0.0;
    fflush(stdout);
  }
  if (needf > 0) {
    /* The nonlinear components of f_i(x) need to be assigned, */
    f[0] = sin(-x[0] - .25) * 1e3 + sin(-x[1] - .25) * 1e3;
    f[1] = sin(x[0] - .25) * 1e3 + sin(x[0] - x[1] - .25) * 1e3;
    f[2] = sin(x[1] - x[0] - .25) * 1e3 + sin(x[1] - .25) * 1e3;
    /* N.B. in this example there is no need to assign for the wholly */
    /* linear components f_4(x) and f_5(x). */
    f[5] = x[2] * (x[2] * x[2]) * 1e-6 + x[3] * (x[3] * x[3]) * 2e-6 / 3.;
  }

  if (needg > 0) {
    /* The derivatives of the function f_i(x) need to be assigned.
     * g[k-1] should be set to partial derivative df_i(x)/dx_j where
     * i = igfun[k-1] and j = igvar[k-1], for k = 1 to LENG.
     */
    g[0] = cos(-x[0] - .25) * -1e3;
    g[1] = cos(-x[1] - .25) * -1e3;
    g[2] = cos(x[0] - .25) * 1e3 + cos(x[0] - x[1] - .25) * 1e3;
    g[3] = cos(x[0] - x[1] - .25) * -1e3;
    g[4] = cos(x[1] - x[0] - .25) * -1e3;
    g[5] = cos(x[1] - x[0] - .25) * 1e3 + cos(x[1] - .25) * 1e3;
    g[6] = x[2] * x[2] * 3e-6;
    g[7] = x[3] * x[3] * 2e-6;
  }

  return;
} /* usrfun */
```

10.2 Program Data

```
Begin nag_opt_sparse_nlp_option_set_file (e04vkc) example options file
* Comment lines like this begin with an asterisk.
* Switch off output of timing information:
Timing level 0
* Allow elastic variables:
Elastic mode 1
* Set the feasibility tolerance:
Feasibility tolerance 1.0E-4
End
```

```
nag_opt_sparse_nlp_option_set_file (e04vkc) Example Program Data
 4   6                : Values of n and nf
 8   8   6   Nag_Cold  : Values of nea, neg, objrow and start

'X1      '  'X2      '  'X3      '  'X4      '  : XNAMES
'NlnCon_1'  'NlnCon_2'  'NlnCon_3'  'LinCon_1'  'LinCon_2'  'Objectiv' : FNAMES

1  3 -1.0E0  : Nonzero elements of sparse matrix A, the linear part of F.
2  4 -1.0E0  : Each row IAFUN(i), JAVAR(i), A(IAFUN(i),JAVAR(i)), i = 1 to nea
4  1 -1.0E0
4  2  1.0E0
5  1  1.0E0
5  2 -1.0E0
6  3  3.0E0
6  4  2.0E0

1  1     : Nonzero row/column structure of G, IGFUN(i), JGVAR(i), i = 1 to neg
1  2
2  1
2  2
3  1
```

```
3  2
6  3
6  4

-0.55E0    0.55E0  : Bounds on the variables, XLOW(i), XUPP(i), for i = 1 to n
-0.55E0    0.55E0
 0.0E0   1200.0E0
 0.0E0   1200.0E0

 -894.8E0 -894.8E0 : Bounds on the functions, FLOW(i), FUPP(i), for i = 1 to nf
 -894.8E0 -894.8E0
-1294.8E0 -1294.8E0
-0.55E0    1.0E25
-0.55E0    1.0E25
-1.0E25    1.0E25

 0.0  0.0  0.0  0.0           : Initial values of X(i), for i = 1 to n
 0    0    0    0             : Initial values of XSTATE(i), for i = 1 to n
 0.0  0.0  0.0  0.0           : Initial values of XMUL(i), for i = 1 to n

 0.0  0.0  0.0  0.0  0.0  0.0 : Initial values of F(i), for i = 1 to nf
 0    0    0    0    0    0   : Initial values of FSTATE(i), for i = 1 to nf
 0.0  0.0  0.0  0.0  0.0  0.0 : Initial values of FMUL(i), for i = 1 to nf
```

10.3 Program Results

```
nag_opt_sparse_nlp_option_set_file (e04vkc) Example Program Results

Option 'Elastic mode' has the value   1.
Option 'Feasibility tolerance' has the value   1.00000e-04.

(User-supplied callback usrfun, first invocation.)
Final objective value =      5126.5
Optimal X =      0.12     -0.40    679.95   1026.07
```