DONLP2 ANSI C

P. SPELLUCCI
TU Darmstadt, FB Mathematik *

July 19, 2001

Abstract

A short description of the peculiarities of `donlp2's` C-version. In any other respect consult the userguide.

1 File structure

The file structure of the code has been simplified compared to the f77-version. there are:

1. the file donlp2.c

   consisting of the optimizer and its subordinate functions and subroutines, including the codes for numerical differentiation

2. the file user_eval.c

   consisting of the interface to the userevaluation code in the so called "block" mode.

3. the file userfu.c

   (used in the makefile) must be created by the user.

The code comes with a set of examples for such files and the "testcommand" simply copies an example to the file userfu.c . This file consists of a main program, for example, in the simplest case

```
main() {
void donlp2(void);

donlp2();

exit(0);
}
```

and the routines

setup0
: which initializes dimensions, parameters, initial point, the descriptive array gunit and gconst (if desired) the minimum which must be given is

[1] authors address:
TU Darmstadt, Dept. of Math. , Schloßgartenstraße 7, D 64289 Darmstadt, Germany.
e-mail: spellucci@mathematik.tu-darmstadt.de

n

nh

ng

tau0

del0

analyt, epsdif

difftype (if analyt=FALSE)

bloc

taubnd (if analyt=FALSE)

epsfcn (if analyt=FALSE)

cold (must be set to FALSE if desired , here. Default is TRUE)

x (=x_initial)

All other variables have reasonable default values.

setup
: is used to override default settings for any of the initializations done by donlp2. you must however not change here the parameters which must be set in setup0, i.e. del0, tau0, x(=x_initial)

solchk
: may contain additional computations with the final result

ef
: evaluates the objective function

egradf
: evaluates the gradient of the objective function body may be empty if analyt=FALSE

eh
: evaluates one of the equality constraints

egradh
: evaluates the gradient of one of the equality constraints

eg
: evaluates one of the inequality constraints

egradg
: evaluates the gradient of one of the inequality constraints.

if bloc=TRUE then ef, egradf,eh,egradh,eg,egradg are never called, hence may have an empty body then.

eval_extern
: is the evaluation code for the bloc-mode. if bloc = TRUE it must fill the arrays fu and fugrad

The number of include-files *.h has been minimized and typically in the user part you need only to access

o8fuco.h
: containing data like dimensions, machine parameters, the descriptive arrays gunit, gconst,llow, lup, the functions counters and the error indicators.

o8para.h
: contains the maximum allowed dimension. If this is changed, you must recompile all object files.

o8cons.h
: contains a set of constants used by the code

o8fint.h contains the parameters needed for the bloc - mode and for numerical differentiation

All other global variables are collected in

o8comm.h
: the names and the meaning of all these variables are the same as for the f77-version, hence the explanation given in the documentation applies without any changes. Warning: o8fuco.h is included in o8comm.h, hence you must not include these files simultaneously.

o8gene.h
: includes all these .h files and some variables necessary in the full sqp mode.

2 Coding of the function subroutines

Coding of the function subroutines is best learned from the examples in the EXAMPLE directory, e.g. the file alkylati.c contains a sufficiently general model. It is included here together with explanations:

```
/* ************************************************************************** */
/*                                 user functions                             */
/* ************************************************************************** */
#include "o8para.h"

main() {
    void donlp2(void);

    donlp2();

    exit(0);
}

/* ************************************************************************** */
/*                              donlp2 standard setup                         */
/* ************************************************************************** */
void setup0(void) {
    #define  X extern
    #include "o8comm.h"
    #undef   X

    static INTEGER  i,j;
    static DOUBLE   xst0[11] = {0.,/* not used : index 0 */
                                1745.e0  ,12.e3  ,11.e1,3048.e0 ,1974.e0,
                                  89.2e0 ,92.8e0 , 8.e0,   3.6e0, 145.e0 };

    /* name is ident of the example/user and can be set at users will       */
    /* the first static character must be alphabetic. 40 characters maximum */

    strcpy(name,"alkylation");  /* problem name*/

    /* x is initial guess and also holds the current solution */
    /* problem dimension n = dim(x), nh = dim(h), ng = dim(g) */

    n      = 10; /* number of variables*/
    nh     = 3;  /* number of equality constraints */
```

```
    ng     = 28; /* number of inequality constraints, including bounds*/
    analyt = TRUE; /*analytical gradients are provided */
    epsdif = 1.e-16;  /* this influences the termination criteria only.
         if analyt=FALSE then epsx>=epsdif^2 and delmin>=epsdif and
         tiny (the indicator for an active constraint in the  qp solver)
         tiny>=2*nr*epsdif, nr the number of active constraints in the
         nonlinear solver) */

    /* epsfcn   = 1.e-16;  function values are accurate to working precision*/
    /* taubnd   = 5.e-6;  bounds may be violated by 5.e-6 during numerical
                          differencing */
    /*  bloc    = TRUE;  indicates bloc-mode*/
    /* difftype = 3;    the 6th order accurate mode*/

    nreset = n;
    /* restart the quasi-newton-update in case of trouble after nreset steps*/
    /* del0 and tau0: see below */

    del0 = 0.2e0;
    tau0 = 1.e0;
    tau  = 0.1e0;
    for (i = 1 ; i <= n ; i++) {
/* initial value */
        x[i] = xst0[i];
    }
    /* gunit-array, see donlp2doc.txt */
/* for this example, we have the objective function (index 0),3 equality
   constraints, 8 general inequality constraints, hence 0<= index<=11 : */
    for (j = 0 ; j <= 11 ; j++) {
        gunit[1][j] = -1;
        gunit[2][j] = 0;
        gunit[3][j] = 0;
    }
    for (j = 12 ; j <= 31 ; j++) {
/* here follows the description of the bound constraints. we have
   10 lower and 10 upper bounds */
        gunit[1][j] = 1;
        if ( j <= 21 ) {
            gunit[2][j] = j-11;  /* index of the variable x */
            gunit[3][j] = 1; /* indicates lower bound *?
        } else {
            gunit[2][j] = j-21;
            gunit[3][j] = -1;   /* indicates upper bound */
        }
    }
    return;
}

/* ********************************************************************** */
/*                                special setup                           */
/* ********************************************************************** */
void setup(void) {
    #define  X extern
    #include "o8comm.h"
```

```
    #undef   X
/* here one can override the standard parameter settings. e.g. setting
   parameters for intermediate output te0,te1,te2,te3 , termination
   parameters (delmin,epsx) etc. */
    return;
}

/* ********************************************************************** */
/*  the user may add additional computations using the computed solution here   */
/* ********************************************************************** */
void solchk(void) {
    #define  X extern
    #include "o8comm.h"
    #undef   X
    #include "o8cons.h"
/* additional evaluations with the final result (with data x,u,....)
   could be done here. all data (primal, dual variables, functions values,
   gradient values, parameters,... ) are located in o8comm.h and can
   hence be accessed */
    return;
}

/* ********************************************************************** */
/*                                objective function                      */
/* ********************************************************************** */
void ef(DOUBLE x[],DOUBLE *fx) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X
/* the objective function. icf is the evaluation counter (in o8fuco.h)*/
    icf = icf+1;
    *fx = 5.04e0*x[1]+.035e0*x[2]+10.e0*x[3]+3.36e0*x[5]-.063e0*x[4]*x[7];

    return;
}

/* ********************************************************************** */
/*                           gradient of objective function               */
/* ********************************************************************** */
void egradf(DOUBLE x[],DOUBLE gradf[]) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X

    static INTEGER  j;
    static DOUBLE   a[11] = {0.,/* not used : index 0 */
                             5.04e0,0.035e0,10.e0,0.e0,3.36e0,
                             0.e0  ,0.e0   , 0.e0,0.e0,0.e0};
/* this is the gradient for f . icgf is the counter for its evaluations */
    icgf = icgf+1;
    for (j = 1 ; j <= 10 ; j++) {
        gradf[j] = a[j];
    }
    gradf[4] = -0.063e0*x[7];
```

```
    gradf[7] = -0.063e0*x[4];

    return;
}

/* ********************************************************************** */
/*                compute the i-th equality constaint, value is hxi       */
/* ********************************************************************** */
void eh(INTEGER i,DOUBLE x[],DOUBLE *hxi) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X
/* there are three equality constraints. hence i varies here form 1 to 3 */
/* cres is the counter for the evaluation of the general constraints */
    cres[i] = cres[i]+1;
    switch (i) {
    case 1:
        *hxi = 1.22e0*x[4]-x[1]-x[5];
        break;
    case 2:
        *hxi = 9.8e4*x[3]/(x[4]*x[9]+1.e3*x[3])-x[6];
        break;
    case 3:
        *hxi = (x[2]+x[5])/x[1]-x[8];
        break;
    }
    return;
}

/* ********************************************************************** */
/*              compute the gradient of the i-th equality constraint      */
/* ********************************************************************** */
void egradh(INTEGER i,DOUBLE x[],DOUBLE gradhi[]) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X

    static INTEGER  j;
    static DOUBLE   t,t1;
/* these are the gradients of the three equality constraints.
   cgres is the counter for the evaluation of these gradients */

    cgres[i] = cgres[i]+1;
    for (j = 1 ; j <= 10 ; j++) {
        gradhi[j] = 0.e0;
    }
    switch (i) {
    case 1:
        gradhi[1] = -1.e0;
        gradhi[4] = 1.22e0;
        gradhi[5] = -1.e0;
        break;
    case 2:
        t         = 9.8e4/(x[4]*x[9]+1.e3*x[3]);
```

```
        t1        = t/(x[4]*x[9]+1.e3*x[3])*x[3];
        gradhi[3] = t-1.e3*t1;
        gradhi[4] = -x[9]*t1;
        gradhi[9] = -x[4]*t1;
        gradhi[6] = -1.e0;
        break;
    case 3:
        gradhi[1] = -(x[2]+x[5])/pow(x[1],2);
        gradhi[2] = 1.e0/x[1];
        gradhi[5] = gradhi[2];
        gradhi[8] = -1.e0;
        break;
    }
    return;
}

/* ********************************************************************** */
/*              compute the i-th inequality constaint, bounds included          */
/* ********************************************************************** */
void eg(INTEGER i,DOUBLE x[],DOUBLE *gxi) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X

    static INTEGER  k;
    static DOUBLE   og[11] = {0.,/* not used : index 0 */
                              2.e3 ,16.e3, 1.2e2,5.e3,  2.e3,
                              93.e0,95.e0,12.e0, 4.e0,162.e0 };

    static DOUBLE   ug[11] = {0.,/* not used : index 0 */
                              1.e-5, 1.e-5,1.e-5,1.e-5,  1.e-5,
                             85.e0, 90.e0 ,3.e0 ,1.2e0,145.e0 };

    static DOUBLE   a = .99e0,b = .9e0,c = 2.01010101010101e-2,
                    d = 2.11111111111111e-1;
    static DOUBLE   t;
/* only constraints which are not bounds are counted as evaluations */
    if ( gunit[1][i+nh] == -1 ) cres[i+nh] = cres[i+nh]+1;
/* there are 8 nonlinear inequalities, 10 lower and 10 upper bounds .
   hence i varies here from 1 to 28. this corresponds to indices
  4 to 31 in the res,gres,gunit  respecively  fu, fugrad arrays */
/* the bounds are at the high end. hence cases 1 to 8 are treated here
   first. since the inequalities appear in pairs even,uneven pairs are
   coded together */

    k = (i+1)/2;

    switch (k) {
    case 1:
        t = 35.82e0-.222e0*x[10]-b*x[9];
        if(k+k == i) t = -t+x[9]*d;
        *gxi = t;
        break;
    case 2:
```

```
        t = -133.e0+3.e0*x[7]-a*x[10];
        if(k+k == i) t = -t+c*x[10];
        *gxi = t;
        break;
    case 3:
        t = 1.12e0*x[1]+.13167e0*x[1]*x[8]-.00667e0*x[1]*pow(x[8],2)-a*x[4];
        if(k+k == i) t = -t+c*x[4];
        *gxi = t;
        break;
    case 4:
        t = 57.425e0+1.098e0*x[8]-.038e0*pow(x[8],2)+.325e0*x[6]-a*x[7];
        if(k+k == i) t = -t+c*x[7];
        *gxi = t;
        break;
    default:
/*  i>8: this is the code for the upper and the lower bounds . It must be given */

        if( i  > 18 ) *gxi = og[i-18]-x[i-18];
        if( i <= 18 ) *gxi = x[i-8]-ug[i-8];
    }
    return;
}

/* ********************************************************************** */
/*              compute the gradient of the i-th inequality constraint          */
/*          not necessary for bounds, but constant gradients must be set        */
/*                      here e.g. using dcopy from a data-field                 */
/* ********************************************************************** */
void egradg(INTEGER i,DOUBLE x[],DOUBLE gradgi[]) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X

    static INTEGER  j,k;
    static DOUBLE   a = .99e0,b = .9e0,c = 1.01010101010101e0,
                    d = 1.11111111111111e0;
/* these are the analytical gradients for the functions from eg */
    for ( j = 1 ; j <= NX ; j++) {
         gradgi[j] = 0.e0;
    }
    k = (i+1)/2;

    switch (k) {
    case 1:
        if ( k+k != i ) {
            gradgi[ 9] = -b;
            gradgi[10] = -.222e0;
        } else {
            gradgi[ 9] =  d;
            gradgi[10] =  .222e0;
        }
        break;
    case 2:
        if ( k+k != i ) {
```

```
            gradgi[ 7] =  3.e0;
            gradgi[10] = -a;
        } else {
            gradgi[ 7] = -3.e0;
            gradgi[10] =  c;
        }
        break;
    case 3:
        gradgi[1] = 1.12e0+.13167e0*x[8]-.00667e0*pow(x[8],2);
        gradgi[4] = -a;
        gradgi[8] = .13167e0*x[1]-.01334e0*x[1]*x[8];
        if( k+k == i ) {
            gradgi[1] = -gradgi[1];
            gradgi[8] = -gradgi[8];
            gradgi[4] = c;
        }
        break;
    case 4:
        gradgi[6] = .325e0;
        gradgi[7] = -a;
        gradgi[8] = 1.098e0-.076e0*x[8];
        if(k+k == i) {
            gradgi[6] = -.325e0;
            gradgi[7] = c;
            gradgi[8] = -gradgi[8];
        }
        break;
    default:
/*  this is for completeness only. since gunit is set, these values
    are never used in the code */
        if( i  > 18 ) gradgi[i-18] = -1.e0;
        if( i <= 18 ) gradgi[i-8]  = 1.e0;
        break;
    }
    return;
}

/* ************************************************************************** */
/*                        user functions (if bloc == TRUE)                    */
/* ************************************************************************** */
void eval_extern(INTEGER mode) {
    #define  X extern
    #include "o8comm.h"
    #include "o8fint.h"
    #undef   X
    #include "o8cons.h"
/*  this is empty here since the bloc mode is not used . Otherwise,
with bloc=TRUE, one could do all computations from above here and store
fx in fu(0)
gradf in fugrad(:,0)
hxi  in fu(1),fu(2),fu(3)
gradhi in fugrad(:,1),...,fugrad(:,3)
gxi  in fu(4),....,fu(31)
gradgi in fugrad(:,4),..,fugrad(:,31).
```

```
*/
    return;
}
```