DONLP2-INTV-DYN USERS GUIDE

P. SPELLUCCI
Technical University at Darmstadt, Department of Mathematics
64289 Darmstadt, Germany
email: spellucci@mathematik.tu-darmstadt.de

1 Copyright

2 General Description

`donlp2-intv-dyn` is a general purpose nonlinear optimizer for continuous variables. It is intended for the minimization of an (in general nonlinear) differentiable real function f subject to (in general nonlinear) inequality and equality constraints . Specifically the problem is accepted in the following form:

$$\begin{aligned} f(x^*) &= \min\{f(x): && x \in \mathcal{S}\} \\ \mathcal{S} &= \{x \in \mathbb{R}^n: && x_u \le x \le x_o\ , \\ & && b_u \le Ax \le b_o\ , \\ & && c_u \le c(x) \le c_o\}\ . \end{aligned}$$

Here A is a matrix of dimension `nlin`×`n` and c is a vectorvalued function of dimension `nonlin`. The lower and upper bounds may have components equal to $-\infty$ resp. ∞ and, if finite, may also be equal, indicating an equality constraint. This applies also for x_u and x_o, $(x_o)_i = (x_u)_i$ indicating a fixed variable. Since there is no internal preprocessor, in that case one explicit equality constraint is used internally. This version uses a dynamic memory allocation and hence needs only as much memory space as is necessary for storing the data for the problems actual dimensions.

2.1 Method employed

The method implemented is a sequential equality constrained quadratic programming method (with an active set technique). The active set is estimated using an error criterion for the Kuhn-Tucker-conditions which is purely local. If linearly dependent gradients of active constraints occur, then the code switches to an alternative usage of a fully regularized mixed constrained subproblem using artificial slack variables with appropriate weights. It uses a slightly modified version of the Pantoja-Mayne update for the Hessian of the Lagrangian, variable dual scaling and an improved Armijo-type stepsize algorithm. Bounds on the variables are treated in a gradient-projection like fashion. Details may be found in the original papers. See chapter on theory.

2.2 System requirements

The software is written in ANSI-C with the exception of two routines `clock()` for giving the processes cpu-time and `ctime` and `time` for writing the run's date and starting time. These usually can be found in

the c-library. If these routines are unavailable, then the user might replace the bodies of clock by "return 0.0", time by *tim=0;" resp. for ctime by an empty body. They have no influence on the functionality of the optimizer.

2.3 Restrictions

Since the algorithm makes no use of sparse matrix techniques, its proper use will be limited to small and medium sized problems with dimensions up to 500 (for the number of unknowns) say. The number of inequality constraints however may be much larger. (e.g. the code did solve the ecker-kupferschmid-marin "robot-design" examples in sijsc 15, 1994, with n=51 unknowns and 3618 inequalities successfully within 400-900 sec's depending on the variation of the problem on a hp9000/735 workstation). There are however no maximally allowed dimensions. In the users part the user must code a function

```
{\tt
void user_init_size(void) {
    #define  X extern
    #include "o8comm.h"
    #include "o8fint.h"
    #include "o8cons.h"
    #undef   X


    /* problem dimension n = dim(x), nlin=number of linear constraints
       nonlin = number of nonlinear constraints  , iterma= number of iterative
       steps aloowed , nstep = number of tries in the backtracking allowed */

    n      = 10;
    nlin   =  5;
    nonlin =  6;
    iterma = 4000;
    nstep = 20;
}
}
```

this is used to allocate memeory.

Unlike the earlier version the gradients of the bound constraints on x are never stored, but of course these constraints count as such. Hence the total number of constraints is `2*(n+nlin+nonlin)` and the number of variables in the SQP-Part can be as large as `2*n+nlin+nonlin` and the constraints there `2*(n+nlin+nonlin)`, namely `n+nlin+nonlin` active constraints plus the same number of slack variables which are bounded from below by zero. However, `gres` has `nlin+nonlin` columns only.

2.4 System dependencies

The code has been successfully tested on the hp9000/7xy , sgi , sun and dec alpha, intel pentium (under linux) systems but should run on any unix-v system without any changes. It has been designed such that changes for other systems are quite simple.
In order to use the code on cray-like systems, reedit donlp2_intv.c, and the include-files (files ending with *.h) changing globally `DOUBLE` to `float`. The code computes the machine precision (called `epsmac` in the code) and the smallest positive machine number (`tolmac`) himself, generating an underflow. This may cause trouble if the system used has the underflow signal not masked. In that case either use appropriate compiler resp. loader switches or replace this computation, located in o8st(), by setting `epsmac` and `tolmac` yourself. E.g. with gcc on an Intel-based machine you must use the compiler options -O3 -ffloat-store

3 Usage

donlp2 is written as a selfcontained system of subprograms. The user has to add his function evaluation programs and a main program, supplying necessary information in a series of userwritten subprograms and in some global variables as described below. an example file is contained in the distribution of the code.

3.1 Problem description

The user has to define the problem through function evaluation routines and the subprogram user_init. This may be done in two ways: method one gives function and gradient evaluation codes individually. In this case the parameter bloc has to be set to FALSE. This is the default. Then the following routines must be supplied:

1 ef(x,&fx) returns fx=$f(x)$ given x as input. arguments: DOUBLE x[],DOUBLE *fx

2 egradf(x,gradf) returns gradf=$\nabla f(x)$ given x as input.
arguments DOUBLE x[], DOUBLE gradf[].

3 econ(type, liste, x, con, err)
returns the value of the $c_i(x),\ i = 1, \ldots,$ nonlin in
con[1],...,con[nonlin] if type==1 (i.e. all nonlin nonlinear constraint functions are evaluated) and
$c_i(x), i =$ liste[1],...,liste[liste[0]] in con[liste[i]] if type==2, given x as input.
arguments: INTEGER type, INTEGER liste[] , DOUBLE x[], DOUBLE con[], LOGICAL err[].
The user can set err[i]=TRUE if for the current x the function c_i is not defined. If this occurs at the initial x, then the code terminates. In all other cases it tries to remedy the situation by reducing the size of the current correction step.

4 econgrad(liste,shift,x,gres) returns $\nabla c_i(x)$ as gres[][liste[i]+shift]
i=liste[1],...,liste[liste[0]] given liste[] and x[] as input. arguments INTEGER liste[] , INTEGER shift, DOUBLE x[], DOUBLE **gres. The code assumes that the gradients of c can be evaluated whenever c itself can.

5 user_init_size has to set actual dimensions, see above.

6 user_init has to set some problem specific data in donlp2's global variables as described below. user_init is the second subprogram called by donlp2 prior to any other computation.

7 setup may set user specific data (not transparent to donlp2) as well as parameters of donlp2 e.g. in order to supersede donlp2's standard parameter setting.
!!! setup is called after user_init and after donlp2's standard initialization, which is done in o8st.

8 solchk may add additional final computations, graphical output using e.g. information from accinf (see below) and other variables. this distribution contains a solchk.c with an empty body.

9 newx Arguments: DOUBLE x[] , DOUBLE u[] , INTEGER itstep , DOUBLE **accinf, LOGICAL *cont. This is a routine which is called in every iteration step giving the user the opportunity to stop computing if for some reason she/he feels that continuing the process would make no sense (maybe the solution is satisfying without the internal termination criteria already met). If cont is set to FALSE, the process stops. accinf contains the history of the optimization in a condensed form, see more below.

The types resp. macros DOUBLE, INTEGER, LOGICAL, TRUE, FALSE, max, min are defined in the file o8para.h , which the user must include in his program part.

The examples directory contains some examples of such a collection of subprograms. The bodies of setup, econ, econgrad and solchk may be empty of course. in that case unconstrained minimization using the BFGS-method is done. The correctness of egradf, econgrad can to some extent be checked by running testgrad.c (in this directory) with the set of user routines . See READTESTGRAD how to use this. The routine newx may be restricted to setting the logical variable cont = TRUE. (the type LOGICAL and the macro TRUE are defined in donlp2's environment.) Alternatively the user might prefer or may be forced to evaluate the problem functions by a black-box external routine. In this case he has to set

```
bloc=TRUE;
```

within user_init and to use a subroutine

```
eval_extern(mode)
```

eval_extern has to take the global vector xtr, which is an external copy of xsc*x, as input and to compute , according to the setting of mode and analyt either fu[i] (i=0:nonlin) or both fu[i] and fugrad[][i]), i=0,nonlin and then return to the optimizer. It also must then set (after successfully finishing its job)

```
valid=TRUE;
```

The meaning of mode for eval_extern is as follows:

1 mode==1 compute only fu, i.e. function values.

2 mode==2 compute function and gradient values fu and fugrad.

Only the objective function f and the nonlinear constraints c are involved, hence indices of fu[i] and fugrad[][i] run from 0 to nonlin. fu[0] corresponds to $f(x)$ and fu[1],...,fu[nonlin] are the values of the nonlinear contraints and the same convention is used for the gradients.

The routines user_init, setup and solchk must be present in this case too, setup and solchk may have an empty body of course.

donlp2 has a built-in feature for doing gradients numerically. This can be used for both methods of problem description. This is controlled by the variables analyt , epsfcn, taubnd and difftype. If analyt==TRUE then donlp2 uses the values from the egrad.. routines or from fugrad, according to the setting of bloc. If analyt=FALSE, then numerical differentiation is done internally using a method depending on difftype.

1 difftype=1: use the ordinary forward difference quotient with discretization stepsize $0.1\texttt{epsfcn}^{1/2}$ componentwise relative.

2 difftype=2: use the symmetric difference quotient with discretization stepsize $0.1\texttt{epsfcn}^{1/3}$ componentwise relative.

3 difftype=3: use a sixth order approximation computing a Richardson extrapolation of three symmetric difference quotient values. This uses a discretization stepsize $0.01\texttt{epsfcn}^{1/7}$. This form of numerical gradients showed in our tests the same precision as analytical gradients.

Here

```
epsfcn
```

is the expected relative precision of the function evaluation. The precision obtained in the gradients is then of the order of $\texttt{epsfcn}^{1/2}$, $\texttt{epsfcn}^{2/3}$, $\texttt{epsfcn}^{6/7}$ respectively. Since numerical differentiation can also occur if variables are on their bounds, the user must supply a further parameter

```
taubnd
```

which gives a positive value by which any active bound constraint may be violated. This occurs using finite differencing with difftype>1. The user is warned that numerical differentiation uses n, $2n$, $6n$ additional function evaluations for a single gradient for `difftype=1,2,3` respectively.

The sequence of routines described above has to be present in any case. (If `eval_extern` is used, then of course the bodies of `ef, egradf, econ, econgrad` are to be empty.) Bound constraints are treated in a special way by `donlp2`. The initial point is corrected automatically to fit the bounds. Thereafter any succeeding point is projected with respect to the bounds.

Within `user_init`, the following data must be given:

1 `name` problem identifier, 40 characters maximum. the leading 8 characters (if any) have to be alphanumeric, the first one alphabetic. nonalphabetic characters are interpreted as `'x'`. `donlp2_intv` computes from this the names of two output-files. See under output below.

2 `x` initial guess (also holds the current solution).

3 `tau0` !!! this parameter should be carefully selected by the user: it gives a (universal) bound describing how much the unscaled penalty-term (the l1-norm of the constraint violation) may deviate from zero, and hence how much any constraint other than a bound can be violated. `donlp2` assumes that within the region described by

$$\sum_i |h_i(x)| - \sum_i \min\{0, g_i(x)\} \ \leq \ \texttt{tau0}$$

all functions may be evaluated safely. Here h_i stands for an equality constraint and g_i for a true inequality constraint of the problem. The initial guess however may violate these requirements. In that case an initial feasibility improvement phase is run by `donlp2` until a point is found, that fits them. This is done by scaling f by `scf=0`, such that the pure unscaled penalty term is minimized. `tau0` may be chosen as large as a user may want. A small `tau0` diminishes the efficiency of `donlp2`, because the iterates then will follow the boundary of the feasible set closely. These remarks do not apply for bound constraints, since `donlp2` treats these by a projection technique. Bounds always remain satisfied. In the example hs114 `tau0=1` works by good luck only. Contrary, a large `tau0` may degrade the reliability of the code (since outdoors the wolfes are howling.)

4 `del0`. In the initial phase of minimization a constraint is considered binding if

$$g_i(x)/\max\{1, ||\nabla g_i(x)||\} \ \leq \ \texttt{del0}.$$

Again g_i symbolizes one of the inequality constraints of the problem. Good values are between 1.d0 and 1.d-2 . If initially `del0` is equal to zero , `del0` is set equal to `tau0` internally. If `del0` is chosen too small, then identification of the correct set of binding constraints may be delayed. Contrary, if `del0` is too large, then the method will often escape to the full regularized SQP method, using individual slack variables for any active constraint, which is quite costly. For well scaled problems `del0=1` is reasonable. however, sometimes it should be much smaller (compare the list of testresults in the accompanying paper).

5 `nreset`. If there are more then `nreset` steps using "small" stepsizes and therefore small corrections, a "restart" of the accumulated quasi-newton-update is tried. `nreset` is internally bounded by 4 from below. Good values are between `n` and `3*n`. In the standard setting `nreset` is bounded by `n`. Override this by using `setup` if desired.

6 `analyt`. Set `analyt=TRUE`, if the gradients are given by analytical expressions. With `analyt=FALSE` donlp2 relaxes its termination criteria using the following variable

7 `epsdif`. `donlp2` assumes relative precision of `epsdif` in the gradients if `analyt` is set to `FALSE`

8 `epsfcn`. This is used only if `analyt==FALSE`. In this case it must give the relative precision of the function evaluation routine. Since precision of the numerical gradients is lower than this, using the method with very crude function values makes no sense.

9 `difftype` Type of numerical differentiation to be used, if any. see above.

10 `silent`. logical. if `TRUE`, neither the result protocol nor the messages protocol is written. `donlp2` returns silently with its results in the appropriate global variables giving no form of output. See the description of these variables below. The user might extract his desired information from these, e.g. in `solchk`. For normal operation silent=FALSE and te0=TRUE is recommended.

11 `taubnd`. Amount by which bounds may be violated if numerical differentiation is used.

12 `cold`. If TRUE, then any information is computed afresh . Otherwise, the penalty-weights and quasi-newton-udates as stored in the globals are used for the current run. This is useful if a parametric problem is to be solved.

13 The parameter `big` must be set, representing infinity. (e.g. big=1.0e20;) Be aware that it is really used for setting the lower and upper bounds for the constraints by default and hence will enter the calculations.

14 The lower bounds $x_u,\ b_u,\ c_u$ (in this order) must be stored in `low[i], i=1,...,n+nlin+nonlin` and similarly the upper bounds $x_o,\ b_o,\ c_o$ in `up[i], i=1,...,n+nlin+nonlin`.

15 The `nlin` rows of the matrix A must be stored as columns in `gres[i][j], i=1,..,n, j=1,..,nlin` . **That means that you have to store the transpose of A in the first columns of gres**.

Default settings are

```
taubnd = 1.e0;
epsfcn = 1.e-16;
epsdif = 1.e-14;
silent = FALSE;
cold   = TRUE;
big    = 1.e20;
intakt = FALSE;
te0    = FALSE;
te1    = FALSE;
te2    = FALSE;
te3    = FALSE;
```

There is an additional routine

```
void newx(DOUBLE x[], DOUBLE u[], INTEGER itstep, DOUBLE **accinf, \\
              LOGICAL cont ) {}
```

It is called after completion of an iterative step and allows the user to control the progress of optimization. Here

`x` is the current solution,

`u` is the vector of Lagrange multipliers,

`itstep` is the step number of the last step

`accinf` is the array of accumulated information defined from [1][] to [`itstep`][]

`cont` to be set TRUE for continuation and FALSE for disruption of the optimization by the user. since all essential data for donlp2 are globals, the user has here the possibility to add action e.g. to modify some settings etc.

The rudimentary form distributed here may be modified according to the users needs.

4 Coding of the function subroutines

Coding of the function subroutines is best learned from the examples in the EXAMPLE directory, e.g. the file hs114_new.c contains a sufficiently general model. It is included here together with explanations: This is the 'alkylation' example from the book of Bracken and McCormick and published as problem 114 in Hock and Schittkowskis book.

the problem is to minimize

$$f(x) = 5.04x_1 + 0.035x_2 + 10x_3 + 3.36x_5 - 0.063x_4x_7$$

subject to

$$\begin{aligned}
h_1(x) &= 1.22x_4 - x_1 - x_5 \;=\; 0 \\
h_2(x) &= 98000x_3/(x_4x_9 + 1000x_3) - x_6 \;=\; 0 \\
h_3(x) &= (x_2 + x_5)/x_1 - x_8 \;=\; 0 \\
g_1(x) &= 35.82 - 0.222x_{10} - bx_9 \;\geq\; 0, \quad b = 0.9 \\
g_2(x) &= -133 + 3x_7 - ax_{10} \;\geq\; 0, \quad a = 0.99 \\
g_3(x) &= -g_1(x) + x_9(1/b - b) \;\geq\; 0, \\
g_4(x) &= -g_2(x) + (1/a - a)x_{10} \;\geq\; 0, \\
g_5(x) &= 1.12x_1 + 0.13167x_1x_8 - 0.00667x_1x_8^2 - ax_4 \;\geq\; 0, \\
g_6(x) &= 57.425 + 1.098x_8 - 0.038x_8^2 + 0.325x_6 - ax_7 \;\geq\; 0, \\
g_7(x) &= -g_5(x) + (1/a - a)x_4 \;\geq\; 0, \\
g_8(x) &= -g_6(x) + (1/a - a)x_7 \;\geq\; 0
\end{aligned}$$

and the bounds

$$\begin{aligned}
0.00001 \;\leq\; x_1 \;&\leq\; 2000 \\
0.00001 \;\leq\; x_2 \;&\leq\; 16000 \\
0.00001 \;\leq\; x_3 \;&\leq\; 120 \\
0.00001 \;\leq\; x_4 \;&\leq\; 5000 \\
0.00001 \;\leq\; x_5 \;&\leq\; 2000 \\
85 \;\leq\; x_6 \;&\leq\; 93 \\
90 \;\leq\; x_7 \;&\leq\; 95 \\
3 \;\leq\; x_8 \;&\leq\; 12 \\
1.2 \;\leq\; x_9 \;&\leq\; 4 \\
145 \;\leq\; x_{10} \;&\leq\; 162\ .
\end{aligned}$$

The problem has all variables bounded from above and below, 5 linear constraints, one of which is an equality constraint and 6 nonlinear constraints, two of which are equalities. This may be coded as follows

```
/* ********************************************************************** */
/*                              user functions                            */
/* ********************************************************************** */
#include "o8para.h"
main() {
    void donlp2(void);
    donlp2();
    exit(0);
}
/* ********************************************************************** */
/*                          donlp2-intv size initialization                          */
/* ********************************************************************** */
void user_init_size(void) {
    #define  X extern
    #include "o8comm.h"
    #include "o8fint.h"
    #include "o8cons.h"
```

```
    #undef   X
    /* problem dimension n = dim(x), nlin=number of linear constraints
       nonlin = number of nonlinear constraints  */
    n      = 10;
    nlin   =  5;
    nonlin =  6;
    iterma = 4000;
    nstep = 20;
}
/* ************************************************************************** */
/*                            donlp2-intv standard setup                      */
/* ************************************************************************** */
void user_init(void) {
    #define  X extern
    #include "o8comm.h"
    #include "o8cons.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 };
    static DOUBLE ugloc[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 ogloc[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   acons = .99e0,bcons = .9e0,ccons = 1.01010101010101e0,
                    dcons = 1.11111111111111e0;
    /* 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");
    /* x is initial guess and also holds the current solution */
    /* problem dimension n = dim(x), nlin=number of linear constraints
       nonlin = number of nonlinear constraints  */
    analyt = TRUE;
    epsdif = 1.e-16;   /* gradients exact to machine precision */
    /* if you want numerical differentiation being done by donlp2 then:*/
    /* epsfcn   = 1.e-16; *//* function values exact to machine precision */
    /* taubnd   = 5.e-6; */
    /* bounds may be violated at most by taubnd in finite differencing */
    /*  bloc    = TRUE; */
    /* if one wants to evaluate all functions  in an independent process */
    /* difftype = 3; *//* the most accurate and most expensive choice */
    nreset = n;
    del0 = 0.2e0;
    tau0 = 1.e0;
    tau  = 0.1e0;
    for (i = 1 ; i <= n ; i++) {
        x[i] = xst0[i];
    }
    /*  set lower and upper bounds */
    big = 1.e20;
    for ( i = 1 ; i <= 10 ; i++ ) { low[i]=ugloc[i];}
    low[11] = -35.82;
    low[13] = 35.82;
    low[12] = 133.0 ;
    low[14] = -133.0;
    low[15]=0.0;
    for ( i = 16 ; i<=21; i++ ) {low[i]=0.0;}
    for ( i = 1 ; i<= 10 ; i++ ) {up[i]=ogloc[i];}
    for ( i = 11 ; i<= 14; i++ ) {up[i]=big;}
    /* 4 linear inequality constraints >=0*/
    up[15]=0.0;   /* linear equality constraint */
    for ( i=16; i<= 19; i++ ) {up[i]=big;}
    /* 4 nonlinear inequality constraints >=0 */
    up[20] = up[21] = 0.0 ;
    /* two nonlinear equalities =0*/
    /* store coefficients of linear constraints directly in gres */
    for ( i = 1 ; i<= 5 ; i++ )
    {
      for  ( j = 1 ; j <= 10 ; j++ )
        {
          gres[j][i] = 0.0 ;
        }
    }
    gres[9][1] = - bcons;
    gres[10][1]= -0.222;
    gres[7][2] = 3.0;
    gres[10][2]= -acons;
    gres[9][3] = 1.0/bcons;
```

```
    gres[10][3]= 0.222;
    gres[7][4] = -3.0;
    gres[10][4]=1.0/acons;
    gres[1][5] = -1.0;
    gres[5][5] = -1.0;
    gres[4][5] = 1.22;
    return;
}
/* ********************************************************************** */
/*                                special setup                           */
/* ********************************************************************** */
void setup(void) {
    #define  X extern
    #include "o8comm.h"
    #undef   X
    te2=TRUE;  /*detailed output of iteration*/
    te3=TRUE;  /* print also gradients of constraints and approximate Hessian of the
                  Lagrangian */
    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"
    return;
}
/* ********************************************************************** */
/*                              objective function                        */
/* ********************************************************************** */
void ef(DOUBLE x[],DOUBLE *fx) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X
    *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};
    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 econ(INTEGER type ,INTEGER liste[], DOUBLE x[],DOUBLE con[],
             LOGICAL err[]) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X
    static INTEGER i,j;
    static INTEGER liste_loc[7];
    static DOUBLE   a = .99e0,b = .9e0,c = 2.01010101010101e-2,
                    d = 2.11111111111111e-1;
    static DOUBLE   t;
/* the six nonlinear constraints are evaluated and stored in con[1],..,con[6] */
/* if type != 1 only a selection is evaluated the indices being taken from    */
/* liste. since we have no evaluation errors here err is never touched        */
    if ( type == 1 )
    {
       liste_loc[0] = 6 ;
       for ( i = 1 ; i<=6 ; i++ ) { liste_loc[i] = i ; }
    }
    else
    {
```

```
       liste_loc[0] = liste[0] ;
       for ( i = 1 ; i<=liste[0] ; i++ ) { liste_loc[i] = liste[i];}
    }
    for ( j = 1 ; j <= liste_loc[0] ; j++ )
    {
      i = liste_loc[j] ;
      switch (i) {
      case 1:
        con[1] =1.12e0*x[1]+.13167e0*x[1]*x[8]-.00667e0*x[1]*pow(x[8],2)-a*x[4];
        continue;
      case 2:
        t = 1.12e0*x[1]+.13167e0*x[1]*x[8]-.00667e0*x[1]*pow(x[8],2)-a*x[4];
        con[2] =  -t+c*x[4];
        continue;
      case 3:
        con[3] = 57.425e0+1.098e0*x[8]-.038e0*pow(x[8],2)+.325e0*x[6]-a*x[7];
        continue;
      case 4:
        t= 57.425e0+1.098e0*x[8]-.038e0*pow(x[8],2)+.325e0*x[6]-a*x[7];
        con[4] = -t+c*x[7];
        continue;
      case 5:
        con[5]  = 9.8e4*x[3]/(x[4]*x[9]+1.e3*x[3])-x[6];
        continue;
      case 6:
        con[6]  = (x[2]+x[5])/x[1]-x[8];
        continue;
      }
    }
    return;
}
/* ********************************************************************** */
/*              compute the gradient of the i-th equality constraint            */
/* ********************************************************************** */
void econgrad(INTEGER liste[] ,INTEGER shift ,  DOUBLE x[],
              DOUBLE **grad) {
    #define  X extern
    #include "o8fuco.h"
    #undef   X
    static INTEGER  i,j,k;
    static DOUBLE   t,t1;
    static INTEGER liste_loc[7];
    static DOUBLE   a = .99e0,b = .9e0,c = 2.01010101010101e-2,
                    d = 2.11111111111111e-1;
    liste_loc[0] = liste[0] ;
    for ( i = 1 ; i<=liste_loc[0] ; i++ ) { liste_loc[i] = liste[i];}
    for ( j = 1 ; j <= liste_loc[0] ; j++ )
    {
      i = liste_loc[j] ;
      for (k = 1 ; k <= 10 ; k++)
      {
           grad[k][i+shift] = 0.e0;
      }
      switch (i) {
      case 1:
/* con[1] =1.12e0*x[1]+.13167e0*x[1]*x[8]-.00667e0*x[1]*pow(x[8],2)-a*x[4];*/
        grad[1][i+shift] = 1.12e0+.13167e0*x[8]-.00667e0*pow(x[8],2);
        grad[4][i+shift] = -a;
        grad[8][i+shift] = .13167e0*x[1]-.01334e0*x[1]*x[8];
        continue;
      case 2:
/*
        t = 1.12e0*x[1]+.13167e0*x[1]*x[8]-.00667e0*x[1]*pow(x[8],2)-a*x[4];
        con[2] =  -t+c*x[4];
*/
        grad[1][i+shift] = -(1.12e0+.13167e0*x[8]-.00667e0*pow(x[8],2));
        grad[8][i+shift] = -(.13167e0*x[1]-.01334e0*x[1]*x[8]);
        grad[4][i+shift] = 1.0/a;
        continue;
      case 3:
/* con[3] = 57.425e0+1.098e0*x[8]-.038e0*pow(x[8],2)+.325e0*x[6]-a*x[7]; */
        grad[6][i+shift] = .325e0;
        grad[7][i+shift] = -a;
        grad[8][i+shift] = 1.098e0-.076e0*x[8];
        continue;
      case 4:
/*
        t= 57.425e0+1.098e0*x[8]-.038e0*pow(x[8],2)+.325e0*x[6]-a*x[7];
        con[4] = -t+c*x[7];
*/
        grad[6][i+shift] = -.325e0;
```

```
        grad[7][i+shift] = 1.0/a;
        grad[8][i+shift] = -(1.098e0-.076e0*x[8]);
        continue;
      case 5:
        t         = 9.8e4/(x[4]*x[9]+1.e3*x[3]);
        t1        = t/(x[4]*x[9]+1.e3*x[3])*x[3];
        grad[3][i+shift] = t-1.e3*t1;
        grad[4][i+shift] = -x[9]*t1;
        grad[9][i+shift] = -x[4]*t1;
        grad[6][i+shift] = -1.e0;
        continue;
       case 6:
        grad[1][i+shift] = -(x[2]+x[5])/pow(x[1],2);
        grad[2][i+shift] = 1.e0/x[1];
        grad[5][i+shift] = 1.e0/x[1];
        grad[8][i+shift] = -1.e0;
        continue;
      }

    }
    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"
    return;
}
```

The code donlp2 computes statistics concerning the number of function and gradient evaluations, taking into account only the general constraints. The relevant counters are global variables which can be accessed. They are located in `o8fuco.h`.

1 `cf` counts the number of function calls of the objective function,

2 `cgf` counts the number of gradient calls of the objective function,

3 `cres(1:nlin+nonlin)` those of the general constraints

4 `cgres(1:nlin+nonlin)` are the counters of the gradient calls for the general constraints. Of course `cgres[i]==1` for `i=1,..,nlin`.

4.1 Additional features

There are additional possibilities to enhance the performance of the code:

1 Parametric problems.
if the user has to solve a problem in a parametric fashion, she/he may do this using `cold=FALSE` in a subsequent call of `user_init` , thereby avoiding the standard initialization of `donlp2` (done in `o8st`). in that case the old quasi-newton-update and the old penalty-weights are used as initial guesses.

2 Parameter settings.
any of the parameters of `donlp2` (e.g. the parameters controlling amount of output `te0, te1, te2, te3`) may be changed at the users will within `setup` by redefining the values of these global variables. this must be done within `setup` whose call follows that of the standard initialization routine `o8st`. in `setup` use `include "o8comm.h"` in order to access these parameters.

3 Error return from function evaluation.
The user has the possibility to prevent `donlp2` from leaving the domain of definition of one of

her(his) functions using the error-indicators either in econ or setting directly ffuerr resp. confuerr[i] to TRUE. The code donlp2 proposes changes of the current solution using some formula

$$x_{new} = x_{current} + \sigma d + \sigma^2 dd$$

with correction vectors d and dd. these directions might well be "downhill" but too large, hence x_{new} leaving the domain of definition of some function involved. The user may check the actual parameter of the function evaluation and set `ffuerr=TRUE` (in the objective function evaluation) or `confuerr[i]=TRUE` in one of the constraint function evaluations and return to the optimizer then. In this case donlp2 tries to avoid the problem by reducing σ. If such an error occurs with the current x, i.e. $\sigma = 0$, then the run is terminated with an appropriate error message.

4 Scaling.
The user has the possibility to prescribe an internal scaling of x, setting the array `xsc` appropriately in `user_init` (must be done there and only there, if any). The initial value given and the function and gradient evaluations however are always in the original unscaled variables. The external variables are then `xsc(i)*x(i)`, with `x(i)` as internal variables. That means that the internal variable is obtained from the external by dividing by `xsc[i]`. The user routines `ef, econ, egradf, econgrad` are always supplied with these external values, hence the user must not use xsc in his evaluation part.

5 Changing termination criteria.
The termination criteria can be changed in various ways. Seven variables enter these criteria (see below), namely

`epsx, delmin, smallw, epsdif, nreset, numsm , epsphi`

The termination criteria used are listed later. `numsm` (the number of allowed successive small relative changes in the penalty function) and `epsphi` may be changed from their default values `max(10,n)` and `1000*epsmac` in order to avoid many steps with only very marginally changing objective function values in the terminal phase of optimization. (Such a situation occurs e.g. for illconditioned cases). The user is warned that being too sloppy with `epsphi` might result in premature termination and solutions far from optimum, if the problem is nonconvex.

5 Output ; Results

donlp2 computes the names of the two files for output using the first 8 characters from `name` . If `name` is shorter than 8 characters `xxx..` is appended to `name` . These files are used for the following: The `name[1:8].pro` file contains the results of the optimization run . Of course these results are also contained in the global variables described below. In case of a failure a short protocol of the complete run is appended, which may be evaluated by a knowledgeable person in order to locate the problem. This post-mortem protocol is obtained from the array `accinf` (with `itstep+1` rows and 33 columns). `itstep` is limited by `iterma` which is defined in `user_init_size`. Output can be of various stages of volume: Normally the final results are written to the pro- and the "special events" to the mes-file. That means te0=te1=te2=te3=FALSE, as given in the head of donlp2. Then the xxxxxxxx.pro -file holds the following:

1. copyright information
2. date and time of run
3. name of problem as defined by the user
4. parameter settings of donlp2 for this run, as left by `setup`.

5. starting value of primal variable
6. termination reason (text)
7. final scaling parameter of primal objective (output of other data is relative to the scaling=1). If this is zero, then `donlp2` failed in the infeasibility improvement phase, where it tries to reduce infeasibility below `tau0`. In that case a larger `tau0` may be tried, or otherwise a different initial guess.
8. $||\nabla f||$
9. error in Lagrangian condition, i.e. $||\nabla_x L||$. Compare with $||\nabla f||$ in order to assess accuracy. In an unconstrained case $||\nabla f|| = ||\nabla_x L||$ of course.
10. primal and dual feasibility errors. For primal the unscaled l1-penalty term and for dual the most negative multiplier, if any.
11. `donlp2`'s cpu-time, accurate to 1/100 second (depending on the precision of `clock()`.
12. optimal value of primal objective ($f(x^*)$).
13. optimal primal value x^*.
14. list of constraint values, max(1,gradient norms) (if evaluated at all) and multipliers. (The norm of a gradient of a constraint which never has been evaluated is set to one. Therefore the norms of the gradients of the active constraints are meaningful only).
15. statistics concerning the evaluation of constraints.
16. condition number estimates for matrix of binding gradients and for quasi-newton update for hessian of the augmented Lagrangian. These are always lower bounds obtained from the diagonal part of the QR-decomposition resp. the diagonal part of the Cholesky-decomposition. Be aware that a large condition number estimate together with `epsx=1.0e-5` might result in a very crude solution. This holds especially true for the Lagrange multipliers which are very sensitive if the constraining manifold is illconditioned.
17. run statistics.

For termination reason (variable `optite`) see below under description of global variables.

If `intakt` is set to `TRUE` in `setup`, any output which appears in the pro-file is written to std-out too.

If `te0` is set true, then a one-line-information is written to std-out for every step of the iteration, giving sufficient information on the progress of solution. This is the following:

1. step number
2. `fx` = value of primal objective
3. `upsi` = value of the unscaled l1-penalty
4. `b2n` = value of l2-norm of error in Lagrange condition (norm of projected gradient of f), i.e. $||\nabla_x L(x, \mu, \lambda)||$.
5. `umi` = smallest negative multiplier or zero
6. `nr` = number of binding constraints
7. `si` -1 = constraints satisfy the regularity condition, 1 = constraints do not satisfy it. In that case a full regularized qp-problem is solved.

If `te1` is set to true, the short protocol accumulated in the array `accinf` is printed at termination. `te1` is set to true automatically in case of unsuccessful termination. The `accinf`-array is a 33-column array, each line of which holds information on an iteration in condensed form, see below .

If `te2` is set to true, intermediate results `x,d`, active constraint values and so on are written to the pro-file. these are:

1. iteration step number
2. `scf` = scaling factor of primal objective
3. `psist` = scaled penalty term at x_initial (current scaling)
4. `psi` = scaled penalty term (current scaling)
5. `upsi` = unscaled l1-penalty term
6. `fxst` = f at x_initial (end of feasibility improvement phase)
7. `fx` = f current
8. `x` = current value of optimization variables (in the internal scaling).
9. permutation of variables applied to make qr-decomposition of matrix of gradients of binding constraints continuous (see coleman and sorensen, math.prog 29)
10. `del` : the current bound for identifying binding constraints. It is decreased and increased dynamically .
11. `b2n0` The projected gradient, based on the current first estimate of the working set (scaled inequality constraints $\leq$ `delmin`), in a weighted norm.
12. `b2n` The final projected Gradient in the euclidean norm, i.e. $||\nabla_x L(x, \mu, \lambda)||$.
13. `gfn` = $||\nabla f(x)||$.
14. list of values of binding constraints and gradient norms (euclidean norm)
15. `diag(r)`= diagonal of r-part of qr-decompostion of matrix of binding gradients
16. `u` = list of multipliers
17. eventually message concerning singularity of the problem and the final result from qp
18. eventually a list of constraints considered for inactivation and multipliers obtained thereafter
19. condition number estimator for matrix of binding constraints and the for the Hessian of the augmented Lagrangian obtained by the Pantoja and Mayne update
20. `d` = direction of descent for the penalty function
21. information on new scaling:
    (a) `scf` scaling for f,
    (b) `clow` : number of penalty decreases
    (c) `eta`: penalty decrease takes place only , if for the new tentative weights
    `(scf*(fxst-fx)+psist-psi)` $\geq$ `clow*eta`.
    `eta` is computed and increased during the run.
    (d) `scalres`: the penalty weights
22. `start unimin`: entry to unidimensional search (stepsize selection)

(a) `phi` = penalty function

(b) `dphi` = its directional derivative along d

(c) `bound upsi = tau0/2`

(d) `psi` : weighted penlty term

(e) `upsi` : unweighted penalty term

(f) `dscal`: scaling of `d`. If the stepsize 1 would change the current `x` too strong, as expressed by

$$||d|| > \beta(||x|| + 1)$$

with the internal parameter `beta` (=4 by default), then `d` is shortened accordingly.

(g) `fx` $f(x)$

(h) `scf` current scaling of f in the penalty function.

(i) `sig` = stepsize tried (σ)

(j) final accepted stepsize

23. list of constraints not binding at x but hit during search

24. kind of matrix updates applied

25. the parameters used in the update

If `te3` is true, the values of the gradients and the approximated Hessians are printed too, provided $n \leq 50$ and `nres=nlin+nonlin`≤ 100.

The *.mes-file is of value if the optimizer ends irregularly. It contains a message for every action which is "abnormal" in the sense of good progress for a regular and well conditioned problem. In most cases special advice is necessary to evaluate it. However, if the user observes many restarts reported in the *.mes file or similarly many calls to the full sqp-method, she(he) should check the problem against bad scaling and redundant constraints!

6 Termination criteria used

`donlp2` has a lot of termination criteria built in, many of which can be controlled by the user. However, an unexperienced user is warned to use too sloppy criteria, since often for nonconvex problems progress in the iteration may be quite irregular. There may occur many steps with seemingly small progress followed by a sudden improvement. The following termination messages may occur, which are explained here in detail:

1. `'constraint evaluation returns error with current point'`
   The user evaluation routine econ (resp. eval_extern) did set an error indicator then asked for a new value and reduction of the trial stepsize down to its minimum did not remove the situation. Usually this means that an inadequate starting point has been provided. Also box constraints may have not been used adequately in order to prevent boundary points of the feasible set there some function is undefined.

2. `'objective evaluation returns error with current point'`
   The same reason as above with respect to the objective function.

3. `'qpsolver: extended qp-problem seemingly infeasible '`
   Theoretically, this is impossible, since the qp problem is constructed such that a feasible solution always exists. Hence this message means that the solution process is severely corrupted by roundoff, most probably a problem of bad scaling which was not overcome by the internal scaling techniques.

4. `'qpsolver: no descent for infeas from qp for tauqp=tau_max'`
   This means that a stationary point of the penalty function has been located which is not feasible. From the theory of the method it follows that the problem is not penalizable, at least in the region of the current point, since the Mangasarian Fromowitz condition, under which global convergence is proved, is the weakest possible for this purpose. Increasing taumax may sometimes remove the situation. (taumax represents a "very big" penalty).

5. `'qpsolver: on exit correction small, infeasible point'`
   same reason as above

6. `'stepsizeselection: computed d from qp not a dir. of descent'`
   same as above or limiting accuracy reached for a singular problem, with termination criteria being too strong. (The method usually tries the outcome d of the qp-solver in spite of problems signaled from the solver as a direction of descent. The final check is done in the stepsize selection subprogram.)

7. `'more than maxit iteration steps'`
   Progress is too slow to meet the termination criteria within the given iteration limit (either maxit or the user defined iterma number of steps) There may be problems with strong infeasibility which let the optimizer stall. In this case try a smaller `tau0`.

8. `'stepsizeselection: no acceptable stepsize in [sigsm,sigla]'`
   This may have a lot of different reasons. The most usual one is simply a programming error in the user supplied (analytical) gradients. Always use `testgrad.c` if you supply your own analytical gradients. It may also be due to termination criteria which are too stringent for the problem at hand, because of evaluation unpreciseness of functions and/or gradients or because of illconditioning of the (projected) Hessian matrix. Often the results are acceptable nevertheless, but the user should check that (e.g. by supplying an appropriate routine solchk)

9. `'small correction from qp, infeasible point'`
   A stationary point of the penalty function which is not feasible for the original problem has been located. Surely the problem is incompatible locally at least. Try some other initial guess.

10. `'kt-conditions satisfied, no further correction computed'`
    means

$$\begin{aligned} ||h(x)||_1 + ||g(x)^-||_1 &\leq \texttt{delmin} \\ ||\lambda^-||_\infty &\leq \texttt{smallw} \\ ||\nabla L(x,\mu,\lambda)|| &\leq \texttt{epsx}(1 + ||\nabla f(x)||) \\ |\lambda^t||g(x)| &\leq \texttt{delmin*smallw*nres} \end{aligned}$$

    Here h stands for the vector of equality constraints of the problem and g for all inequality constraints. μ and λ are the computed Lagrangian multipliers.

11. `'computed correction small, regular case '`
    means

$$\begin{aligned} ||d|| &\leq (||x|| + \texttt{epsx})\texttt{epsx} \\ ||\nabla L(x,\mu,\lambda)|| &\leq \texttt{epsx}(1 + ||\nabla f(x)||) \\ ||h(x)||_1 + ||g(x)^-||_1 &\leq \texttt{delmin} \\ ||\lambda^-||_\infty &\leq \texttt{smallw} \end{aligned}$$

    where d is the computed correction for the current solution x.

12. `'stepsizeselection: x almost feasible, dir. deriv. very small'`
means that the directional derivative has become so small that progress cannot be expected. More precisely

$$d\phi(x;d) \;\geq\; -100(|\phi(x)|+1) * \texttt{epsmac}$$

where ϕ is the current penalty function. This is the usual termination reason for illconditioned problems .

13. `'kt-conditions (relaxed) satisfied, singular point'`
means that we are at a point x where the matrix the constraint normals is illconditioned or singular and the following conditions are met, which describe a relaxed form of the kkt-conditions:

$$\begin{aligned} ||h(x)||_1 + ||g(x)^-||_1 &\leq \texttt{delmin*nres} \\ ||\nabla L(x,\mu,\lambda)|| &\leq 100 \times \texttt{epsx}(1+||\nabla f(x)||) \\ ||sl||_1 &\leq \texttt{delmin*nres} \end{aligned}$$

where sl is the vector of articial slacks introduced to make the qp always consistent. observe that dual feasibility and complementary slackness hold automatically for the current iterate, because we are solving a full qp subproblem.

14. `'very slow primal progress, singular or illconditoned problem'`
means the occurence of the following conditon for at least `n` consecutive steps:

$$\begin{aligned} ||h(x)||_1 + ||g(x)^-||_1 &\leq \texttt{delmin*nres} \\ ||\lambda^-||_\infty &\leq \texttt{smallw} \\ |\lambda^t||g(x)| &\leq \texttt{delmin*smallw*nres} \\ ||\nabla L(x,\mu,\lambda)|| &\leq \texttt{10*epsx}(1+||\nabla f(x)||) \\ |f(x)-f(x_{prev})| &\leq (|f(x)|+1)\texttt{eps} \end{aligned}$$

where `eps` is computed from the machine precision `epsmac` and the variables `epsx` and `epsdif` as follows:

```
if ( analyt ) then
  eps=min(epsx,sqrt(epsmac))
else
  eps=epsdif
  if ( epsx .lt. epsdif**2 ) epsx=epsdif**2
endif
eps=max(epsmac*1000,min(1.d-3,eps))
```

15. `'more than nreset small corrections in x '`
means that for more than `nreset` consecutive steps there were only componentwise small changes in the solution x as given by

$$|x_{prev,i} - x_i| \;\leq\; (|x_i| + 0.01) \times \texttt{epsx}\ , \quad i = 1,\ldots,n$$

16. `'correction from qp very small, almost feasible, singular '`
means that the gradients in the working set are linearly dependent or almost so, such that a full regularized `qp` is solved, with the outcome of a direction d satisfying

$$||d|| \;\leq\; 0.01(\min\{1,||x||\} + \texttt{epsx})\texttt{epsx}\ .$$

this could effect changes in $||x||$ less than $0.01 \times$ `epsx` (relative) at most, hence we terminate since this may be highly inefficient. It may occur in a problem there the second order sufficiency condition is not satisfied and the matrix of gradients of binding constraints is singular or very illconditioned.

17. `'numsm small differences in penalty function,terminate'`
    For `numsm` consecutive steps there were only small changes in the penalty function (without the other termination criteria satisfied) as given by

$$\phi(x_{prev}) - \phi(x) \ \leq \ \texttt{epsphi}(s|f(x)| + \psi(x))$$

    where ϕ is the current penaltyfunction, ψ the current penalty term and s the current scaling of the objective function.

18. `'user required termination'`. The `newx` subprogram returned the value `continue=FALSE`, signaling that the user wished to terminate the run irrespective of the fufillment of the termination criteria.

7 Restrictions

1) !!!!! names beginning with `o8` are reserved by `donlp2` and must not be used otherwise within the users load module.

2) !!!!! the parameters of `o8para.h` must not be changed by the user without having donlp2_intv.c recompiled

8 File structure

There are:

the file donlp2.c

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

the file user_eval.c

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

the file userfu.c

(used in the makefile) must be created by the user. The `testsingle` and `testcommand` create `userfu.c` by copying a file from the `examples` directory to userfu.c.

o8fuco.h
: containing data like dimensions, machine parameters, the descriptive arrays llow, lup, the functions counters and the error indicators. Normally this will be included in the users function subprograms.

o8para.h
: contains the definition of the types resp. macros `INTEGER, DOUBLE, LOGICAL, TRUE, FALSE, max, min`.

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 meaning of all these variables is explained below.

Warning: o8fuco.h is included in o8comm.h, hence you must not include these files simultaneously. See below.

The code comes with a set of examples for user function evaluation 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

user_init_size
: which initializes dimensions `n, nlin, nonlin, iterma, nstep`

user_init which sets parameters, initial point etc. The minimum information which must be given here is

- 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)
- low and up
- A if nlin>0. The rows of A must be stored columnwise in gres[][i], i=1,..,nlin.

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 user_init, i.e. `n, nlin, nonlin, 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

econ
: Evaluates the nonlinear constraints

econgrad
: Evaluates the gradients of the nonlinear constraints as necessary

newx
: Decides whether to continue the optimization or not. Normally it only sets `continue=TRUE;` and returns.

If bloc=TRUE then ef, egradf,econ,econgrad are never called, hence may have an empty body then.

9 Theory

Mathematical background of the method may be found in the two papers

P. Spellucci: *An SQP method for general nonlinear programs using only equality constrained subproblems.* Math. Prog. 82, (1998), 413–448
P. Spellucci: *A new technique for inconsistent problems in the SQP method.* Math. Meth. of Oper. Res. 47, (1998), 355–400. (published by Physica Verlag, Heidelberg, Germany).
The basic concept is the following: given a guess x_{cur} of the solution, we first guess the set of binding constraints using all equality constraints and those inequality constraints with values $\leq$ `delmin` (remember that internally the problem is in the form $h(x) = 0,\ g(x) \geq 0$.) Using this working set we estimate the Lagrangian multipliers using a least squares approach for $\nabla_x L(x, \mu, \lambda) = 0$. With the primal-dual guess obtained we estimate the distance to a Kuhn-Tucker point in a fashion as used by Facchinei, Kanzow et. al. but indeed already contained in the authors 1980 paper "Han's method without using QP". This distance estimation now defines our current `del` and we add to the working set all inequality constraints with

$$g_i(x)/\max\{1, ||\nabla g_i(x)||\} \ \leq \ \texttt{del}$$

and repeat the estimation of the dual variables. If the working set is singular, we switch to a full SQP step using additional slack variables with associated penalty weights for all constraints in the working set. Otherwise we will use a single linear equation to define a direction of descent for the l1-penalty function. A subset of the inequality constraints with negative multiplier estimates is selected for (indirect) inactivation. This inactivation is done by modifying the descent direction appropriately. Having obtained a descent direction by one of these two branches, we perform an appropriately constructed line search, getting x_{next}. The method is enhanced by projection with respect to the bounds, by the use of a quadratic arc rather than a line to overcome the Maratos effect and a modified BFGS update for approximating the Hessian of the Lagrangian.

10 Description of donlp2's global variables

The globals used by `donlp2` are listed here. (They may be accessed from `o8comm.h` and its further included `*.h` files).

10.1 Information pertaining of the post mortem dump

accinf contains information about intermediate results useful for a postmortem analysis .

```
accinf a c c u m u l a t e d   i n f o r m a t i o n
on iteration sequence
0: not used
```

```
 1: step-nr
 2: f(x_k) current value of objective (zero in feasibility improvement
           phase (-1)  )
 3: scf    internal scaling of objective (zero in phase -1)
 4: psi    the weighted penalty-term
 5: upsi   the unweighted penalty-term (l1-norm of constraint vector)
 6: del_k_1 bound for currently active constraints
 7: b2n0   weighted l2-norm of projected gradient, based on constraints in level delmin
           and below
 8: b2n    l2-norm of projected gradient based on del_k_1
 9: nr     number of binding constraints
10: sing   if 1, the binding constraints don't satisfy the regularity condition
11: umin   infinity norm of negative part of inequalities multipliers.
12:        presently not used
13: cond_r condition number of diagonal part of qr-decomp. of normalized
           gradients of binding constraints
14: cond_h condition number of diagonal of cholesky-factor of updated full Hessian
15: scf0   the relative damping of tangential component if upsi>tau0/2
16: xnorm  l2-norm of x
17: dnorm  l2-norm of d (correction from eqp/qp -subproblem, unscaled)
18: phase  -1 : infeasibility improvement phase, 0: initial optimization
           1: binding constraints unchanged , 2: d small, maratos correction
           in use
19: c_k    number of decreases of penalty weights
20: wmax   infinity norm of weights
21: sig_k  stepsize from unidimensional minimization (backtracking)
22: cfincr number of objective function evaluations for stepsize-algorithm
23: dirder directional derivative of penalty-function along d (scaled)
24: dscal  scaling factor for d
25: cosphi cos of arc between d and d_previous. if larger theta , sig larger than
           one (up to sigla) is tried if {\tt sig=1} gives already sufficient
           decrease.
26: violis[0] number of constraints not binding at x but hit during line search
27:        type of update for Hessian: 1 normal p&m-bfgs-update, 0 update suppressed,
           -1 restart with scaled unit matrix , 2 standard bfgs, 3 bfgs modified
           by powells device
28: ny_k or tk:   modification factor for damping the projector in the bfgs
             resp. pantoja-mayne update  respectively
29: 1-my_k or  xsik modification factor for damping the
         quasi-newton-relation in bfgs.
         for unmodified bfgs ny_k should be larger than updmy0 (near one)
           and 1-my_k equal one  resp. the pantoja-mayne
           regularization term .
30: qpterm 0, if sing=-1, termination indicator of the qp-solver otherwise,
             namely
           1: successful, -1: tau becomes larger than tauqp without slack-
           variables becoming sufficiently small .
           -2: infeasible qp-problem (theoretically impossible)
31: tauqp: weight of slack-variables in qp-solver
32: infeas l1-norm of slack-variables in qp-solver
```

10.2 Information pertaining of the termination reason

optite is the termination parameter. optite ranges from -10 to 8, a negative value indicating some irregular event. optite+11 corresponds one the following messages:

```
'constraint evaluation returns error with current point',
'objective evaluation returns error with current point',
'qpsolver: working set singular in dual extended qp ',
'qpsolver: extended qp-problem seemingly infeasible ',
'qpsolver: no descent for infeas from qp for tau=tau_max',
'qpsolver: on exit correction small, infeasible point',
'stepsizeselection: computed d from qp not a dir. of descent',
'more than maxit iteration steps',
'stepsizeselection: no acceptable stepsize in [sigsm,sigla]',
'small correction from qp,  infeasible point',
'kt-conditions satisfied, no further correction computed',
'computed correction small, regular case ',
'stepsizeselection: x almost feasible, dir. deriv. very small',
'kt-conditions (relaxed) satisfied, singular point',
'very slow primal progress, singular or illconditoned problem',
'more than nreset small corrections in x ',
'correction from qp very small, almost feasible, singular ',
'numsm small differences in penalty function,terminate'
'user required termination'
```

a "good" termination is case 8,9,10.(value 0,1,2) In the cases 12-15 the final precision might be quite poor, since in the singular case donlp2 reduces its requirements automatically by multiplying `epsx` by 100. But nevertheless quite often the solution is very good also in these cases. Hence the user must decide whether the result is acceptable in these cases.

10.3 Information pertaining of the current point

runtim : the processes cpu-time

phase : phase -1 : infeasibility improvement phase, 0: initial optimization 1: binding constraints unchanged , 2: d small, maratos correction in use

x the current point

x0 the previous point

x1 the tentative new point

xst the true starting point (as left by `o8st`)

The same name convention is used for the names upsi , psi , fx , sig, *norm

fx function value (fx0, fx1, fxst)

psi the scaled penalty-term (psi0, psi1, psist)

upsi unscaled penalty term (upsi0, upsi1, upsist)

phi phi=scf*f+psi the penalty function.

xmin during the stepsize-increase-phase the current best point (with values sigmin, upsim, psimin, fmin, phimin, resmin)

sig the current step-size

dirder directional derivative of phi along d

cosphi =cos(arc(d,d0))

dscal scaling of d (if too long, resulting in a too large change of x)

d direction of change

dd second order (Maratos) correction

difx difx = x_new-x_old

b2n the projected gradient

b2n0 projected gradient based on a reduced minimal working set , but in a scaled system using the cholesky factor of the current Hessian approximation.

gradf the gradient of objective function at the current point

qgf internal weighted orthogonal transform of gradf

gphi1

gphi0 gradient of Lagrangian at the new and the old point using the old estimation of the active set and inequality constraints with positive multipliers only (remember that donlp2 treats active inequalities as equality constraints).

gres gradients of general constraints, row zero is used for the sign, i.e. gres contains the rows of A and the gradients of c and the sign is positive if the lower bound is active, otherwise negative. Observe that some of these columns may contain invalid information since only the gradients in the working set are evaluated at the current point.

gresn gresn[i]=max(1,norm of gradient of general constraint i)

10.4 Information pertaining of the QR-decomposition

The following variables give complete information on the QR-decomposition of the matrix made up from the gradients of the currently binding constraints.

perm current valid row permutation making the qr-decomposition a continuous function of the input (as proposed by Sorensen)

perm1 new permutation

colno column permutation using column pivoting in the qr-decomposition. colno is used in the routine o8qpdu with double length.

rank rank of working set

qr matrix containing information of the householder-decomposition of the working set in linpack manner. since we may have `n+nlin+nonlin` constraints in the working set in the worst case, the r-part may be trapezoidal.

diag diagonal of the r-part.

betaq factor for reflection normals.

cscal the column scaling.

colle column length for the qp-problem (extended columns)

10.5 Information pertaining of the constraints

These are additional variables used in function evaluation.

val val[i]=true, if gradient of constraint i has been evaluated at the new point i=0 corresponds to f

llow

lup llow and lup are computed by `donlp2` using low and up. llow[i]==TRUE if there is a finite lower bound for x[i] lup(i)==TRUE if there is a finite upper bound for x[i]

10.6 Some additional parameters

intakt intakt==TRUE: give output to pro-file and to console

te0 te0 == TRUE : give one-line-output for every step on console

te1 te1== TRUE : give post-mortem-dump of accumulated information in accinf

te2 te2 == TRUE : give detailed protocol of iteration

te3 te3 == TRUE : also print the gradients and the approximated Hessian

singul singul == TRUE : the working set is singular

ident ident == TRUE : we try a modified step with the same point

eqres eqres == TRUE : the working set stays fixed

silent silent== TRUE : give no output at all (user must evaluate information from the globals himself)

analyt analyt == TRUE : analytical gradients are provided (either in `egradf, econgrad` or in `fugrad`.) Assume full precision in the gradients. Otherwise the user must set epsdif accordingly. If `analyt==FALSE`, then donlp2 does its own numerical gradients, depending on the setting of `difftype, taubnd epsfcn` and automatically reduces its precision requirements.

cold `cold == TRUE` : initialize weights, Hessian and so on, otherwise use them unchanged in the first step of the current run. In a parametric problem in the very first run there must hold `cold ==FALSE` of course. Warning: the code himself cannot check whether `cold` is used properly. Therefore `cold` is set to `FALSE` prior to calling `user_init`.

10.7 Information pertaining of the quasi newton update

a The quasi-Newton-update in cholesky-factors: the upper triangle contains the current update and the strict lower triangle + diag0 the previous one. (This is of course not A from the problem description!)

matsc scaling for identity to be used in case of restart (computed internally) in a fashion similar to that proposed by Shanno.

10.8 Information pertaining of the working set

violis list of constraints hit during current linesearch

aalist complete indices of the working set (including bounds)

alist list of active general constraints

clist list of nonlinear constraints/ constraints gradients which remain to be evaluated

o8bind for constraints binding at x (violated or at zero) o8bind[i]=1 (=0 otherwise)

o8bind0 the same for x0

10.9 Information pertaining of the multipliers and the penalty weights

u current mulipliers estimates ,

u0 previous multipliers estimates,

w penalty weights ,

w1 tentative new weights

res* constraint values , res0 = previous constraint values, res1 = same at the tentative point , resst = constraint values at starting point . Remember that there are always 2*(n+nlin+nonlin) formal constraints some of which may however never be active (depending on the values of low and up).

scf current scaling of f .

scf0 damping factor for tangential part of d if infeasibility > tau0/2.

yu yu internal variable, stores multipliers for working set.

slack slack values in qp-problem .

infeas infeas = one-norm of slack .

work work array.

10.10 Information regarding dimension parameters

n n=dim(x)

nlin number of general linear constraints,

nonlin number of nonlinear constraints =dim(c),

nres nres=nlin+nonlin+n,

nr nr=dim(working set) (may be larger than n !!!!!)

There must hold n $\leq$ `NX`, nlin+nonlin $\leq$ `NRESM`, where `NX` and `NRESM` are set in `o8para.h`.

10.11 Information regarding machine parameters and the stepsize for differencing

`epsmac` machine precision, computed internally.

`tolmac` smallest positive machine number (roughly approximated) computed automatically by donlp2. This computation causes an underflow which may not be tolerated on some systems. Also check the automatic computation of epsmac. This must be the precision of data storage, not that of the internal processor, which may be higher on some systems. Maybe you must suppress some compiler optimization in order to force that (although in the code there is a provision to avoid this inconvenience)

`deldif` deldif is a suitable difference stepsize for the automatic numerical differentiation routine supplied with this distribution. It is computed automatically depending on `epsmac`.

10.12 Information regarding performance and tuning of the code

iterma The maximum number of steps allowed. (30*n is a good value from experience). Must be $\leq$ `MAXIT` as set in `o8para.h`.

del* del = current delta, del0 = maximum of delta , del01=del0/10 ,

delmin constraints are considered as sufficiently satisfied if their violation is less than delmin (absolute). An inequality constraint is always considered as active if its value is less than delmin (internally the inequality constraints have the form $g_i(x) \geq 0$.)

tau0 upper bound for the unscaled penalty-term , i.e. the one-norm of the constraint violation, allowed during the main iteration phase. The initial value may violate this condition. In this case a feasibility improvement phase is run first.

tau tau gives a weight between descent for f and infeasibility and is also used as a safety parameter for chosing the penalty weigths. It can be chosen larger zero at will, but useful values are between 0.1 and 1. The smaller tau, the more may f be scaled down. Tau is also used as an additive increase for the penalty-weights. Therefore it should not be chosen too large, since that degrades the performance of the code.

ny ny increase factor for penalty-weights when computed from the multipliers using
w[i] = ny * u[i] + tau ;

smalld : Use second order correction if $||d|| \leq$ smalld and the working set stays fixed.

smallw Multipliers absolutely smaller than smallw are considered as zero. Hence the dual feasibility violation max norm may be as large as smallw.

rho The working set considered as singular if the condition number of the diagonal of the r-part of its qr-decomposition (after scaling the gradients to one) is larger than 1/rho.

rho1 If the condition number of quasi-Newton-update of Hessian becomes larger than 1/rho1, a restart with a scaled unit matrix is performed.

eta level required for descent if weights are to be diminished, computed automatically

epsx termination parameter. successful termination is indicated if (roughly speaking) the kuhn-tucker-criteria are satisfied within an error of epsx

epsphi Variable entering the termination criterion for "small progress" in the penalty function, see under termination criteria.

c1d Indirect inactivation of an inequality contraint occurs if a multiplier*c1d is less than the negative of the norm of the scaled projected gradient at the current point. (see theory)

scfmax The internal automatic scaling of f is bounded by 1/scfmax from below and scfmax from above. this scaling aims at one multiplier at least being in the order of magnitude of one.

tauqp The penalty-parameter for the slacks in the qp-problem.

taufac increase factor for tauqp.

taumax maximum allowable tauqp.

10.13 Information regarding the stepsize algorithm

alpha alpha=smallest reduction factor for stepsize (presently 0.1).,

beta feasible increase factor for change of x-norm adding the correction d. If d is too long, it is rescaled such that the change in x cannot be larger than beta*(norm(x)+1) if the stepsize is one.

theta If theta $\leq$ cos(arc(d,d0)) then stepsizes greater than 1 are tried for d, where d0=previous correction.

sigsm smallest stepsize allowed.

sigla largest stepsize allowed in the increase phase of sigma. This may result in changes of x larger than those preset by the choice of beta. These large changes however occur only if they give better progress.

delta The armijo-parameter.

stptrm The termination indicator for the stepsize-algorithm.

delta1 armijo-parameter used for the reduction of infeasibility test if `upsi>tau0/2`.

stmaxl min(sigla, maximum of stepsize for which the point on the ray x+sigma*d, projected to the bounds, changes) that is the largest useful sigma allowed.

10.14 Information regarding computing statistics

cf Number of calls of f.

cgf Number of calls of grad(f).

cfincr Number of function values used in the current stepsize-selection.

cres cres[i]= number of calls to individual constraints (bounds not counted)

cgres The same for the gradients.

This function evaluation statistics is now done by donlp2 himself.

10.15 Additional information to be set by the user function evaluation program

ffuerr Error indicator for evaluating f.

cfuerr cfuerr[i] error indicator for evaluating a nonlinear constraining function $c_i(x)$.
These are always initialized by FALSE.

10.16 Information regarding the construction of the penalty function

clow Number of times the weights have been dimished.

lastdw Last step number with weights diminished.

lastup Last step number with weights increased.

lastch Last step number with weights changed.

10.17 Further variables

name This is the problem identifier, to be set in user_init. The output file name is constructed from it. If name is undefined, then the output file names are xxxxxxxx.*

epsdif is the precision to be set if the user himself computes the gradients numerically. epsdif should be the estimated precision in the gradients. otherwise use epsdif=machep.

ug, og the scaled lower and upper bounds for x (internal), i.e. ug[i]=low[i]/xsc[i], og[i]=up[i]/xsc[i],i=1,...,n.

nreset The allowed number of successive steps for which there is no or little progress. Triggers reinitialization of the quasi-Newton-update. It should never be larger than 2*n or 3*n, but usually 10 is a reasonable value for it.

numsm The number of allowed consecutive small changes in the penalty function.

xst The initial vector (possibly after establishing the bounds).

10.18 Information for interchange between the user and the optimizer in the case of a block evaluation

The following global variables serve as an interface for transmitting and receiving information from a black-box evaluation routine which a user might wish to use:

bloc If bloc==TRUE, then it is assumed that function evaluation takes place at once in an external routine and that user_eval has been called (which in turn calls eval_extern(mode)) before calling for evaluation of functions (esf, esgradf,escon, escongrad). The latter then simply consists in copying data from the fu*** variables to donlp2's own data-areas.

corr corr is set to true by donlp2, if the initial x does not satisfy the bound constraints. x is modified in this case.

difftype difftype==1,2,3 numerical differentiation by the ordinary forward differences, by central differences or by richardson-extrapolation of order 6, requiring n, 2n , 6n additional function evaluations respectively.

epsfcn is the assumed precision of the function evaluation routine, to be set by the user.

taubnd : amount by which bound constraints may be violated during finite differencing, set by the user. This does not occur for difftype==1.

xtr : The current design x rescaled for external use.

fu[i] : The vector of function values in the order i==0 objective function, i==1,...,nonlin nonlinear constraints. The same convention is used for fugrad[][i] for the corresponding gradients. With bloc==TRUE and analyt=FALSE only fu must be computed, using xtr as input.

11 Results obtained in testing donlp 2

These can be found in the papers describing the underlying theory and in the distribution containing the original f77 version of the code.

12 Some advice

How to proceed:

1) Check whether the dimensioning is appropriate for your memory size. You need about

   (2*n+nlin+nonlin)*(6+2*(2*n+nlin+nonlin))+(68+3n+2*nstep)*(n+nlin+nonlin)+40*n+33*iterma

   double precision numbers to store.

2) If you work on a unix-system, edit the makefile (you possibly must modify the compilerflags).

3) Having written your problem-description, first compile and link it with testgrad.c in order to check your analytical gradients if you have supplied these. you should supply analytical expressions whenever this is possible. The numerical differentiation feature will normally degrade reliability and efficiency. With `difftype=3` , which of course is quite costly, all testcases but one could be solved to almost the same precision as with analytical gradients. However the costs are very high ($6n$ function evaluations for a single gradient). In testgrad, if you use half of the machine precision as discretization stepsize then you should obtain half of the machine precision as accuracy in the numerical gradients, that is the relative errors should all be in the order of 1.0e-8 in the IEEE754 doube precision. This assumes that the function values themselves have full machine precision.

4) !!! in writing your example, remember you must completely specify your problem, such that the evaluation of any function is well defined in the box given by the bound constraints. e.g. if you have to use log , $\sqrt{}$, x^y with y real or double, /(....) when you must add the appropriate lower bounds $\geq \epsilon > 0$ (or $\leq -\epsilon < 0$ for denominators) for the expressions occuring there.

5) !!! Many machines nowadays use ieee-arithmetic and have floating-errors disabled. optimizing NaN's of course is useless. If possible, use linker- or compiler switches which enable error reporting of floating point errors (overflow, divide by zero, integer overflow) in order to detect such behaviour. Exponent underflow errors must be disabled however. Such an underflow will necessarily occur in computing tolmac. If you cannot disable the underflow error signal you must remove the relevant part of the code and set tolmac yourself.

6) compile your application programs and link with donlp2.o. If you work on a unix-system, you may copy the suite of function evaluation routines to userfu.c and type "make exe" like done in testsingle and testcommand.

7) !!! if `donlp2` behaves "strange", proceed as follows: use

```
   subroutine setup
   #include "o8comm.h"
   te2=TRUE;
/* the following for small n, nlin+nonlin */
   te3=TRUE;
   .......
   return
   end
```

   After runnning the testcase, grep for 'NaN' or 'nan' in the *.pro-file. If these are present, you can be sure that there is an error in your coding of the example of the kind indicated above.