NLPAPI: An API to Nonlinear Programming Problems. User's Guide

Michael E. Henderson
IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598
mhender@watson.ibm.com

June 30, 2003

2.1 Functions

The problem consists of functions $F(x)$, which are scalar valued functions of the problem variables x. In the problem definition there are two functions, the objective function $O(x)$ and constraints $c_i(x)$.

The data variable allows the user to associate a data block with the objective, that is passed to the functions when they are evaluated. The freedata routine is called when the problem is free'd, so that the data block can be released. The array v lists the nv problem variables on which the objective (or constraint, since the same form is used for those) depends.This allows a simple form of sparsity).

The correspondance between the arguments of F and the problem variable is handled in the same way as for the expression. So to set the objective this way would require code:

```
v[0]=0;v[1]=10;v[2]=9;
rc=NLPSetObjective(P,name,3,v,F,dF,ddF,data,freedata);
```

The basic procedure for defining a function this way is add the required number of groups, then to set the group scales and group functions, the

```
rc=NLPSetSimpleBounds(P,i,l,u);
rc=NLPSetLowerSimpleBound(P,i,l);
rc=NLPSetUpperSimpleBound(P,i,u);
```

3.1.1 The Objective

Each problem has an objective function. This can be set with the `NLPSetObjective` or `NLPSetObjectiveByString` routine, or can be built as the sum of a number of groups.

(or constraint, since the same form is used for those) depends.This allows a simple form of sparsity).

dF evaluates the partial derivatives of f, and has an additional integer argument (the first argument), which indicates which partial derivative to return. ddF evaluates the second partial derivatives, and has two additional integer arguments (the first two).

Alternatively, the user can define the objective by means of a string containing an expression:

```
NLPSetObjectiveByString(P,name,nv,v,
    "[x1,x2,x3]",
    "(x1-x2)**2A1(e,xc)1(ti2A1(x2-1(ti02)1()1*)12/*)19+"()1(31)1(5e))1(1*)
```

r(m)stdlbs

```
int v[3];
double (*F)(int,double*,void*);
double (*dF)(int,int,double*,void*);
double (*ddF)(int,int,int,double*,void*);
void *data;
```

```
NLPSetInequalityConstraintGroupFunction
NLPSetInequalityConstraintGroupScale
NLPSetInequalityConstraintGroupA
NLPSetInequalityConstraintGroupB
NLPAddNonlinearElementToInequalityConstraintGroup
```

Inequality constraints can be evaluated using the routines:

```
int c;
double o;
NLVector v,g;
NLMatrix H;

o=NLPEvaluateInequalityConstraint(P,c,v);

g=NLCreate...Vecton(...);
```

```
      void *data;
      void (*freedata)(void*);

      nv=3;v[0]=3;v[1]=10;v[2]=9;
      l=1.;u=10.;
      rc=NLPAddEqualityConstraint(P,nam2,nv,v,F,dF,ddF,
                                                  data,freedata);
```

The data

Inequalities are sometimes dealt with by introducing extra variables called slacks. That is,

$$l \leq f(x) \leq u$$

is replaced by an equality constraint and simple bounds on the slack –

$$f(x) - s$$

```
NLPSetObjectiveGroupFunction(g,gf);
```

di□erent GroupFunctions), and freedata is a routine that is 1, toled when the GroupFunction is freed.

```
ef=NLCreateElementFunctionWithInitialHessian(P,"etype",
                                             n,R,F,dF,ddF,
                                             data,freedata,
                                             ddF0);

ef=NLCreateElementFunctionByString(P,"etype",n,R,
                                             "[x,y,z,w]",
                                             "x**2+y**2-z*w");
```

Here, n is the number of element variables, R the range transformation (or NULL), F, dF, and ddF are routines which evalute F and its derivatives (ddF may be NULL). If ddF is NULL, ddF0 gives an initial guess at the Hessian

3.1.8 Matrices

releases the storage. It calls NLFree.. for all of the groups, element functions, and so on which are stored in the problem. When the user creates one of these data structures a "reference count" associated with it is set to "1". When the problem stores a pointer to the data structure the reference count is increased by one. The "NLFree..." routines decreases the reference count by one and if the count is zero, releases the memory used by the data structure. For example:

```
g=NLCreateGroupFunction(...);      ref count = 1
NLPSetObjectiveGroupFunction(...); ref count = 2
NLFreeGroupFunction(...);          ref count = 1 not yet
```

The severity is 4, 8 or 12, the Routine is the routine which issued the error, and the line and file give the line of source code where it was issued. The

2 Example

We will develop the code for c9eating and solving HS65. HS65 is the problem:

minimize
(*x*

```
NLPSetSimpleBounds(P,1,-4.5,4.5);
```

function as the group function, but element functions, unlike groups, which take a scalar argument, take a vector as argument.

First we include the API prototypes:

```
#include <NLPAPI.h>
```

Note that for `NLCreateElementFunction` the number of internal variables is passed (i.e. the actual number of unknowns used by the function), as well as a m54(h3ange)-349(tra)1(ns)-1(fo)1(rmatio)1(n",)-355(whic)27(h)-350(b)28(y)-350(default

```
rc=NLPSetObjectiveGroupA(P,group,a);
rc=NLPSetObjectiveGroupB(P,group,5.);
NLFreeVector(a);
```

Next come bounds on the variables: