Multiobjective Proximal Bundle Method for Nonconvex Nonsmooth Optimization: Fortran Subroutine MPBNGC 2.0

Marko M. Mäkelä

Department of Mathematical Information Technology
P.O. Box 35 (Agora), FIN-40014 University of Jyväskylä, Finland
makela@mit.jyu.fi

Abstract: We consider a new implementation version 2.0 of MPBNGC Fortran subroutine. MPBNGC is based on the multiobjective proximal bundle method for nonlinearly constrained nonconvex and nonsmooth optimization. In the new version the quadratic subproblem is solved by PLQDF1 subroutine, which is an realization of the dual range space quadratic programming method for minimax approximation with linear constraints implemented by Lukšan. PLQDF1 replaces QPDF4 subroutine of Kiwiel realizing the dual active-set quadratic programming algorithm.

Keywords: Nonsmooth optimization, bundle methods, quadratic programming, multiobjective optimization, software.

1991 Mathematics Subject Classification: 90C25, 65K05.

1. Introduction

Nonsmooth (nondifferentiable) optimization problems arise in very many fields of applications, for example, in economics [Outrata et al., 1998], mechanics [Moreau et al., 1988], engineering [Mistakidis and Stavroulakis, 1998] or in optimal control [Mäkelä and Neittaanmäki, 1992]. On the other hand, instead of one criterion the applications typically have several, often conflicting objectives [Miettinen, 1999].

The source of nonsmoothness may be the objective function itself, for example economics piecewise linear tax models or the constraint functions like in optimal control problems governed by partial differential systems. On the other hand the solution procedures based on decomposition methods may also lead to the nonsmooth problem. We may add that there also exist so called stiff problems which are smooth analytically but nonsmooth numerically.

There are also several approaches to solve nonsmooth optimization problems. The usage of smooth gradient-based methods for nonsmooth problems is a simple approach but may lead to a failure in convergence, in optimality test or in gradient approximation (see [Lemaréchal, 1989]). On the other hand, the direct methods, for example, the method of Hooke and Jeeves (see, e.g., [Bazaraa and Shetty, 1979]) employing no derivative information, become inefficient when the size of the problem is growing. Different kind of regularization techniques introduced, for instance, in [Haslinger and Neittaanmäki, 1996] may give satisfactory results in some cases but are not, in general, as efficient as the direct nonsmooth approach, as was noticed in [Mäkelä and Neittaanmäki, 1992].

Caused by the facts listed above the need of efficient methods for nonsmooth problems (possibly with several objectives) is evident and widely accepted (see [Mäkelä, 2002]). However, the commercial software for mathematical programming, like subroutine libraries and most of other optimization packages, are capable to solve only smooth optimization problems. At the moment the pioneering solvers M1FC1 and M2FC1 (see [Lemaréchal and Bancora Imbert, 1985]) derived in [Lemaréchal, 1975], BT package (see [Outrata et al., 1991]) derived in [Schramm and Zowe, 1992], optimization systems UFO derived in [Lukšan et al., 2000] and NOA described in [Kiwiel, 1991], and subroutine library NSOLIB including the code MPBNGC (see [Mäkelä, 1993]) are probably the only packages for nonsmooth optimization.

In this work we consider a new implementation version 2.0 of MPBNGC Fortran subroutine. The code MPBNGC combines the multiobjective linearization technique (see [Wang, 1989]) and proximal bundle method (see [Kiwiel, 1990], [Mäkelä and Neittaanmäki, 1992]) for nonlinearly constrained nonconvex optimization. MPBNGC has been used as a local optimizer in the WWW-NIMBUS system (see [Miettinen and Mäkelä, 2000]) realizing the interactive multiobjective optimization method NIMBUS (see [Miettinen and Mäkelä, 1995, 2002]). In the new version of MPBNGC the quadratic subproblem is solved by PLQDF1 subroutine, which is an realization of the dual range space quadratic programming method for minimax approximation with linear constraints [Lukšan, 1984]. The code PLQDF1 replaces QPDF4 subroutine realizing the dual active-set quadratic programming algorithm [Kiwiel, 1986].

The paper is organized as follows. In Section 2 we give the problem formulation and remain some basic results from nonsmooth anlysis. In Section 3 we shortly describe the multiobjective proximal bundle method with some theoretical optimality results. The sections 4 and 5 are devoted to the operation and the numerical testing of MPBNGC, respectively; an example of the user-provided Fortran codes including the parameter description is presented and one numerical example with the well-known test problem from literature is reported. In Section 6 we give some numerical evidence of the reliability of the new version of MPBNGC by comparing it with the previous version in some large-scale single objective test problems.

2. Preliminaries

Let us consider a nonsmooth multiobjective optimization problem of the form

$$\begin{array}{ll} \text{minimize} & f(x) = (f_1(x), \ldots, f_k(x))^T \\ \text{subject to} & g(x) = (g_1(x), \ldots, g_m(x))^{\mathrm{T}} \leq 0, \end{array} \tag{1}$$

where the objective functions $f_i \colon \mathbb{R}^n \to \mathbb{R}$ and the constraint functions $g_i \colon \mathbb{R}^n \to \mathbb{R}$ are supposed to be locally Lipschitz continuous (see, e.g., [Mäkelä and Neittaanmäki, 1992]). The word "minimize" means that we want to minimize all the objective functions simultaneously.

Optimality is here understood in the sense of Pareto. A feasible point $x^* \in \mathbb{R}^n$ is *Pareto optimal* if none of the components of f can be decreased without increasing at least one of the other components, that is, there does not exist another feasible point x such that $f_i(x) \leq f_i(x^*)$ for all $i = 1, \ldots, k$, and $f_j(x) < f_j(x^*)$ for at least one $j \in \{1, \ldots, k\}$. A criterion vector $f(x^*)$ is said to be Pareto optimal if x^* is Pareto optimal.

In addition to Pareto optimality, we also employ a more general concept, weak Pareto optimality. A feasible point $x^* \in \mathbb{R}^n$ is *weakly Pareto optimal* if there does not exist any other feasible point where every component of f has a smaller value, that is, there does not exist another feasible point x such that $f_i(x) < f_i(x^*)$ for all $i = 1, \ldots, k$. A criterion vector $f(x^*)$ is said to be weakly Pareto optimal if x^* is weakly Pareto optimal. Every Pareto optimal solution is weakly Pareto optimal.

For a locally Lipschitz continuous function $f_i \colon \mathbb{R}^n \to \mathbb{R}$, a *subdifferential* at a point x is defined (see [Clarke, 1983]) as

$$\partial f_i(x) = \operatorname{conv}\{\xi \in \mathbb{R}^n \mid \xi = \lim_{j\to\infty} \nabla f_i(x^j),\ x^j \to x,\ \nabla f_i(x^j) \text{ exists}\},$$

which is a nonempty, convex and compact subset of $\mathbb{R}^n$ (see, e.g., [Mäkelä and Neittaanmäki, 1992]).

Theorem 1. *If the function $f_i \colon \mathbb{R}^n \to \mathbb{R}$ is locally Lipschitz continuous at $x^* \in \mathbb{R}^n$ and attains its local minimum at x^*, then*

$$0 \in \partial f_i(x^*).$$

If the function f_i is convex, then the condition is also sufficient and the minimum is global.

For this and other preliminary results of convex and nonsmooth analysis, we refer to [Rockafellar, 1982], [Clarke, 1983] and [Mäkelä and Neittaanmäki, 1992]. Now we can present necessary conditions for weak Pareto optimality.

Theorem 2. *Let the objective and the constraint functions of the problem (1) be locally Lipschitz continuous at the feasible point $x^* \in \mathbb{R}^n$. A necessary condition for x^* to be a weakly Pareto optimal solution of the problem (1) is that there exist multipliers $0 \leq \lambda \in \mathbb{R}^k$ and $0 \leq \mu \in \mathbb{R}^m$ for which $(\lambda, \mu) \neq (0, 0)$ such that*

$$\begin{aligned} &(a) \quad 0 \in \sum_{i=1}^{k} \lambda_i \partial f_i(x^*) + \sum_{j=1}^{m} \mu_j \partial g_j(x^*) \\ &(b) \quad \mu_j g_j(x^*) = 0 \quad \textit{for all} \quad j = 1, \ldots, m. \end{aligned}$$

For the proof we refer to [Miettinen and Mäkelä, 1995].

A feasible point $x^* \in \mathbb{R}^n$ is called a *substationary point* if it satisfies the necessary optimality conditions of Theorem 2.

If some constraint qualification is valid, we can obtain the necessary optimality conditions in a form where some of the components of λ is strictly positive. If the problem (1) is convex then it is said to satisfy the *Slater constraint qualification* if there exists some x^* with $g_j(x^*) < 0$ for all $j = 1, \ldots, m$.

Theorem 3. *Let the problem (1) be convex (implying that all the objective and the constraint functions are locally Lipschitz continuous). A sufficient condition for a feasible point $x^* \in \mathbb{R}^n$ to be a weakly Pareto optimal solution of the problem (1) is that there exist multipliers $0 \leq \lambda \in \mathbb{R}^k$ with $\lambda \neq 0$ and $0 \leq \mu \in \mathbb{R}^m$ such that*

$$\begin{aligned} (a) \quad & 0 \in \sum_{i=1}^{k} \lambda_i \partial f_i(x^*) + \sum_{j=1}^{m} \mu_j \partial g_j(x^*) \\ (b) \quad & \mu_j g_j(x^*) = 0 \quad \text{for all} \quad j = 1, \ldots, m. \end{aligned}$$

If the Slater constraint qualification is valid, then the above mentioned conditions are also necessary for weak Pareto optimality.

The proof can be found in [Miettinen and Mäkelä, 1995].

3. Multiobjective Proximal Bundle Method

Next we briefly sketch the multiobjective proximal bundle method. The original proximal bundle method of [Kiwiel, 1990] is the most advanced version of the bundle family for nonsmooth single objective optimization. It has been generalized to handle nonconvex and constrained problems in [Mäkelä and Neittaanmäki, 1992]. The following multiobjective proximal bundle method is an extension into a multiobjective case. The strategy of handling several objective functions is based on the linearization technique presented in [Kiwiel, 1985(a)] and [Wang, 1989].

Let us first consider an *improvement function* $H \colon \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ defined by

$$H(x, y) = \max\{f_i(x) - f_i(y),\ g_j(x) \mid i = 1, \ldots, k,\ j = 1, \ldots, m\}.$$

Now we obtain the following connection between the improvement function and the problem (1).

Theorem 4. *A necessary condition for the point $x^* \in \mathbb{R}^n$ to be a weakly Pareto optimal solution of the problem (1) is that*

$$x^* = \operatorname*{argmin}_{x \in \mathbb{R}^n} H(x, x^*). \tag{2}$$

If the problem (1) is convex and the Slater constraint qualification is satisfied, then the condition (2) is also sufficient.

For the proof we refer to [Miettinen and Mäkelä, 1995].

Let x^h be the current approximation to the solution of (1) at the iteration h. Then, by Theorem 4, we seek for the search direction d^h as a solution of the unconstrained optimization problem

$$\begin{aligned} &\text{minimize} && H(x^h + d, x^h) \\ &\text{subject to} && d \in \mathbb{R}^n. \end{aligned} \tag{3}$$

Since (3) is still a nonsmooth problem, we must approximate it somehow.

Let us assume for a moment that the problem is convex. We suppose that, at the iteration h besides the current iteration point x^h, we have some auxiliary points $y^j \in \mathbb{R}^n$ from the past iterations and subgradients $\xi^j_{f_i} \in \partial f_i(y^j)$ for $j \in J^h = \{1, \dots, h\}$, $i = 1, \dots, k$, and $\xi^j_{g_l} \in \partial g_l(y^j)$ for $j \in J^h$, $l = 1, \dots, m$. We linearize the objective and the constraint functions at the point y^j by

$$\begin{aligned} \bar{f}_{i,j}(x) &= f_i(y^j) + (\xi^j_{f_i})^{\mathrm{T}}(x - y^j) \quad \text{for all} \quad i = 1, \dots, k,\ j \in J^h, \quad \text{and} \\ \bar{g}_{l,j}(x) &= g_l(y^j) + (\xi^j_{g_l})^{\mathrm{T}}(x - y^j) \quad \text{for all} \quad l = 1, \dots, m,\ j \in J^h. \end{aligned}$$

Now we can define a convex piecewise linear approximation to the improvement function by

$$\hat{H}^h(x) = \max\{\bar{f}_{i,j}(x) - f_i(x^h),\ \bar{g}_{l,j}(x) \mid i = 1, \dots, k,\ l = 1, \dots, m,\ j \in J^h\}$$

and we get an approximation to (3) by

$$(4) \qquad \begin{aligned} &\text{minimize} \quad \hat{H}^h(x^h + d) + \tfrac{1}{2}u^h\|d\|^2 \\ &\text{subject to} \quad d \in \mathbb{R}^n, \end{aligned}$$

where $u^h > 0$ is some weighting parameter. The penalty term $\frac{1}{2}u^h\|d\|^2$ is added to guarantee that there exists a solution to (4) and to keep the approximation local enough.

Notice that (4) is still a nonsmooth problem, but due to its min-max-nature it is equivalent to the following (differentiable) quadratic problem

$$(5) \qquad \begin{aligned} &\underset{(d,v)\in\mathbb{R}^{n+1}}{\text{minimize}} \quad v + \tfrac{1}{2}u^h\|d\|^2 \\ &\text{subject to} \quad -\alpha^h_{f_i,j} + (\xi^j_{f_i})^T d \le v,\ i = 1, \dots, k,\ j \in J^h \\ &\qquad\qquad\quad -\alpha^h_{g_l,j} + (\xi^j_{g_l})^T d \le v,\ l = 1, \dots, m,\ j \in J^h, \end{aligned}$$

where

$$\begin{aligned} \alpha^h_{f_i,j} &= f_i(x^h) - \bar{f}_{i,j}(x^h), \quad i = 1, \dots, k,\ j \in J^h, \quad \text{and} \\ \alpha^h_{g_l,j} &= -\bar{g}_{l,j}(x), \quad l = 1, \dots, m,\ j \in J^h \end{aligned}$$

are so-called linearization errors. In nonconvex cases, we replace the linearization errors by so-called subgradient locality measures

$$\begin{aligned} \beta^h_{f_i,j} &= \max\,[|\alpha^h_{f_i,j}|, \gamma_{f_i}\|x^h - y^j\|^2] \\ \beta^h_{g_l,j} &= \max\,[|\alpha^h_{g_l,j}|, \gamma_{g_l}\|x^h - y^j\|^2], \end{aligned}$$

where $\gamma_{f_i} \ge 0$ for $i = 1, \dots, k$ and $\gamma_{g_l} \ge 0$ for $l = 1, \dots, m$, are so-called distance measure parameters ($\gamma_{f_i} = 0$ if f_i is convex and $\gamma_{g_l} = 0$ if g_l is convex).

Let (d^h, v^h) be a solution of (5). We perform the following two-point line search strategy, which will detect discontinuities in the gradients of the objective functions. We assume that $m_L \in (0, \frac{1}{2})$, $m_R \in (m_L, 1)$ and $\bar{t} \in (0, 1]$ are some fixed line search parameters. First, we search for the largest number $t_L^h \in [0,1]$ such that

$$\max\{f_i(x^h + t_L^h d^h) - f_i(x^h) \mid i = 1, \dots, k\} \leq m_L t_L^h v^h, \quad \text{and}$$
$$\max\{g_l(x^h + t_L^h d^h) \mid l = 1, \dots, m\} \leq 0.$$

If $t_L^h \geq \bar{t}$, we take a *long serious step*:

$$x^{h+1} = x^h + t_L^h d^h \quad \text{and} \quad y^{h+1} = x^{h+1},$$

if $0 < t_L^h < \bar{t}$, then we take a *short serious step*:

$$x^{h+1} = x^h + t_L^h d^h \quad \text{and} \quad y^{h+1} = x^h + t_R^h d^h$$

and if $t_L^h = 0$, we take a *null step*:

$$x^{h+1} = x^h \quad \text{and} \quad y^{h+1} = x^h + t_R^h d^h,$$

where $t_R^h > t_L^h$ is such that

$$-\beta_{f_i,h+1}^{h+1} + (\xi_{f_i}^{h+1})^{\mathrm{T}} d^h \geq m_R v^h.$$

We use the line search algorithm of [Mäkelä and Neittaanmäki, 1992] to produce the step-sizes t_L^h and t_R^h. The iteration is terminated when $-\frac{1}{2}v^h < \varepsilon_s$, where $\varepsilon_s > 0$ is an accuracy parameter supplied by the user. The subgradient aggregation strategy due to [Kiwiel, 1985(b)] is used to bound the storage requirements (i.e., the size of the index set J^h) and a modification of the weight updating algorithm of [Kiwiel, 1990] is used to update the weight u^h.

Next we consider some optimality results concerning the solutions produced by the multiobjective proximal bundle method.

Theorem 8. *Let the multiobjective optimization problem (1) be convex and Slater's constraint qualification be satisfied. If the multiobjective proximal bundle method stops with a finite number of iterations, then the solution is weakly Pareto optimal. On the other hand, any accumulation point of an infinite sequence of solutions generated by the multiobjective proximal bundle method is weakly Pareto optimal.*

For the proof we refer to [Kiwiel, 1985(a)] and [Wang, 1989].

If the convexity assumption is not satisfied, we obtain somewhat weaker results about substationary points. In addition, we have to assume that the objective and the constraint functions are weakly semismooth. A function $f_i \colon \mathbb{R}^n \to \mathbb{R}$ is said to be *weakly semismooth* if the directional derivative $f_i'(x,d) = \lim_{t\downarrow 0}(f_i(x + td) - f_i(x))/t$ exists for all x and d, and $f_i'(x,d) = \lim_{t\downarrow 0} \xi(x+td)^T d$, where $\xi(x+td) \in \partial f_i(x+td)$.

Theorem 9. *Let the functions of the multiobjective optimization problem (1) be weakly semismooth. If multiobjective proximal bundle method stops with a finite number of iterations, then the solution is a substationary point. On the other hand, any accumulation point of an infinite sequence of solutions generated by the multiobjective proximal bundle method is a substationary point.*

For the proof, see [Wang, 1989] and the references therein.

4. Specification of MPBNGC 2.0

The code MPBNGC is an Fortran 77 implementation of the multiobjective proximal bundle method described in previous chapter for nonsmooth, nonconvex and generally constrained optimization problem

$$
(6) \qquad \begin{cases} \text{minimize} & f_1(x), \ldots, f_m(x) \\ \text{subject to} & f_j(x) \le 0, \quad j = m+1, \ldots, m+m_g \\ & Cx \le b, \\ \text{and} & b_l \le x \le b_u, \end{cases}
$$

where the functions $f_j : \mathbb{R}^n \to \mathbb{R}$ for $j = 1, \ldots m + m_g$ are supposed to be locally Lipschitz continuous, C is $m_c \times n$ constraint matrix, b is m_c-vector of right-hand-sides, and b_l, b_u are n-vectors of lower and upper bounds, respectively.

In the new version 2.0 of MPBNGC the dual of the quadratic subproblem (5) is solved by PLQDF1 subroutine, which is an realization of the dual range space quadratic programming method for minimax approximation with linear constraints [Lukšan, 1984]. The code PLQDF1 replaces QPDF4 subroutine used in previous versions of MPBNGC. QPDF4 is an implementation of the dual active-set quadratic programming algorithm [Kiwiel, 1986].

The communication with MPNNGC has to be done under the following guidelines. The user has to provide the main program calling MPBNGC and the subroutine FASG for function and subgradient calculation. MPBNGC must be linked together with PLQDF1. The heading of MPBNGC subroutine is the following.

```
C*****************************************************************
      SUBROUTINE MPBNGC (N, X, MF, MG, F, FASG, MC, C, B, RL
     &                   LMAX, GAM, EPS, FEAS, JMAX, NITER,
     &                   NFASG, NOUT, IPRINT, IERR, BL, BU,
     &                   IWORK, LIWORK, WORK, LWORK, IUSER, USER)
C*****************************************************************
      INTEGER            N, MF, MG, MC, LMAX, JMAX, NITER,
     &                   NFASG, NOUT, IPRINT, IERR, LIWORK,
     &                   LWORK, IWORK(LIWORK), IUSER(*)
      DOUBLE PRECISION   RL, EPS, FEAS,
     &                   X(N), F(MF+MG), C(MC+1,N),
     &                   B(MC+1), GAM(MF+1), BL(N),
     &                   BU(N), WORK(LWORK), USER(*)
      EXTERNAL           FASG
```

Parameter description of MPBNGC 2.0 is the following.

N – **Integer** *Input*

On entry: The number of variables.

Constraint: N $\geq$ 1.

X(N) – **Double precision** array *Input/Output*

On entry: The initial approximation to the solution.
On exit: The best calculated approximation to the solution.
Constraint: X must be feasible with respect to constraints.

MF – **Integer** *Input*
On entry: The number of objective functions.
Constraint: MF ≥ 1.

MG – **Integer** *Input*
On entry: The number of general constraint functions.
Constraint: MG ≥ 0.

F(MF+MG) – **Double precision** array *Output*
On exit: The objective and general constraint function values at the solution.

FASG – **Subroutine**, supplied by the user *External Procedure*
Must calculate the function values and single subgradients of the objective and general constraint functions. FASG must be declared as EXTERNAL in the mainprogram from which MPBNGC is called. The specification of FASG is the following.

```
C*****************************************************
   SUBROUTINE FASG (N, X, MM, F, G, IERR, IUSER, USER)
C*****************************************************
   INTEGER          N, MM, IERR, IUSER(*)
   DOUBLE PRECISION X(N), F(MM), G(N,MM), USER(*)
```

N – **Integer** *Input*
On entry: The number of variables.

X(N) – **Double precision** array *Input*
On entry: The current iteration point.

MM – **Integer** *Input*
On entry: The total number of objective and general constraint functions, i.e. MM = MF + MG.

F(MM) – **Double precision** array *Output*
On exit: The objective and general constraint function values at the current iteration point.

G(N,MM) – **Double precision** array *Output*
On exit: The matrix containing in its columns the subgradients of the objective and general constraint functions at the current iteration point.

IERR – **Integer** *Output*
On exit: The failure parameter. If there exist problems in function or subgradient calculations, set IERR = 8.

IUSER(*) – **Integer** array *Workspace*

USER(*) – **Double precision** array *Workspace*

MC – **Integer** *Input*

On entry: The number of linear constraints.

Constraint: MC $\geq$ 0.

C(MC+1,N) – **Double precision** array *Input*

On entry: The coefficient matrix of linear constraints. If MC = 0, then C is not referenced.

B(MC+1) – **Double precision** array *Input*

On entry: The right hand side vector linear constraints. If MC = 0, then B is not referenced.

RL – **Double precision** *Input*

On entry: The line search parameter m_L.

Constraint: 0 < RL < 0.5.

LMAX – **Integer** *Input*

On entry: The upper bound for function and subgradient calculations per iteration.

Constraint: LMAX $\geq$ 1.

GAM(MP1) – **Double precision** array *Input*

On entry: The vector of distance measure parameters γ. The dimension MP1 is defined by

$$\text{MP1} = \begin{cases} \text{MF} + 1, & \text{if MG} > 0; \\ \text{MF}, & \text{if MG} = 0. \end{cases}$$

The first M components are devoted to the corresponding objective functions and the (possible) last component is common for all the general constraint functions. Set GAM(i) = 0 if the corresponding ith function is convex.

Constraint: GAM(i) ≥ 0, $i = 1, \ldots,$MP1.

EPS – **Double precision** *Input*

On entry: The final accuracy tolerance.

Constraint: EPS > 0.

FEAS – **Double precision** *Input*

On entry: The tolerance for constraint feasibility.

Constraint: FEAS > 0.

JMAX – **Integer** *Input*

On entry: The upper bound for the size of the bundle, i.e. for stored subgradients.

Constraint: JMAX $\geq$ 2.

NITER – **Integer** *Input/Output*

On entry: The upper bound for number of iterations.

On exit: The number of used iterations.

Constraint: NITER $\geq$ 1.

NOUT – **Integer** *Input*

On entry: The output unit number.

NFASG – **Integer** *Input/Output*

On entry: The upper bound for the FASG calls.

On exit: The number of used function and subgradient calls.

Constraint: NFASG ≥ 2.

IPRINT – **Integer** *Input*

On entry: The printout control parameter.

-1 :	No printout.
0 :	Only the error messages.
1 :	The final values of the objective functions.
2 :	The whole final solution.
3 :	At each iteration the values of the objective functions.
4 :	At each iteration the whole solution.

IERR – **Integer** *Output*

On exit: The failure parameter.

0 :	Everything is OK.
1 :	The number of calls of FASG = NFASG.
2 :	The number of iterations = NITER.
3 :	Invalid input parameters.
4 :	Not enough working space.
5 :	Failure in quadratic solver.
6 :	The starting point is not feasible.
7 :	Failure in attaining the demanded accuracy.
8 :	Failure in function or subgradient calculations (assigned by the user).

BL(N) – **Double precision** array *Input*

On entry: The vector of lower bounds for X.

Constraint: BL(i) $\leq$ BU(i), $i = 1,\ldots,$N.

BU(N) – **Double precision** array *Input*

On entry: The vector of upper bounds for X.

Constraint: BU(i) $\geq$ BL(i), $i = 1,\ldots,$N.

IWORK(LIWORK) – **Integer** array *Workspace*

LIWORK – **Integer** *Input*

On entry: The dimension of the array IWORK.

Constraint: LIWORK $\geq$ 2×(MP1×(JMAX+1)+MC+N).

WORK(LWORK) – **Double precision** array *Workspace*

LWORK – **Integer** *Input*

On entry: The dimension of the array WORK.

Constraint: LWORK $\geq$ MP1×(6×JMAX+10)+JMAX+ 5×MC+MG+MF+18
+N×(N+2×MP1×JMAX+2×MP1 +2×MC+2×MG+2×MF+31)/2.

IUSER(*) – **Integer** array *Workspace*

USER(*) – **Double precision** array *Workspace*

5. Problem Example

5.1 Problem Formulation.

To demonstrate the solution process in details we shall solve by MPBNGC the following test problem

$$\begin{cases} \text{minimize} & f_1(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 \\ & f_2(x) = \max\{x_1^2 + (x_2 - 1)^2 + x_2 - 1, -x_1^2 - (x_2 - 1)^2 + x_2 + 1\} \\ \text{subject to} & (x_1 - 1)^2 + (x_2 - 1)^2 - 1 \le 0 \\ & x_1 + x_2 \le 1 \\ & 0 \le x_1, x_2 \le 1. \end{cases}$$

Thus we want to solve a two dimensional bicriteria optimization problem subject to one nonlinear and one linear constraint in the unit square. The first objective function is the classical (smooth) Rosenbrock's nonconvex banana function, which attains its unconstrained minimum at $(1, 1)$ (nonfeasible in this example). The second one is the nonsmooth and highly nonconvex test function Crescent, which has the global minimum at $(0, 0)$ (nonfeasible). The starting point was chosen to be $(1, 0)$, where both of the constraint functions and also the upper bound for x_1 and lower bound for x_2 are active.

The used MPBNGC parameters were fixed as follows. The stopping criteria ε_s (EPS) was set to be 10^{-5}, i.e. we demand approximately 5 right digits in the final solution. The value 0.01 for the line search parameter m_L (RL) was noticed to work well. The upper bound for the stored subgradients $J_{\max}$ (JMAX) was chosen to be 5. Due to the nonconvexity of the objective functions the values 0.3, 0.6 and 0.0 were used for the distance measure parameters γ_{f_1} (GAM(1)), γ_{f_2} (GAM(2)) and γ_{f_3} (GAM(3)), respectively. In order not to restrict the solution process via limiting the function or subgradient evaluations each of the parameters LMAX, NITER and NFASG was chosen to be 100.

5.2. Program Text.

In the following we present the Fortran main program calling MPBNGC and the subroutine FASG calculating the function and subgradient values.

```
C*******************************************************************
C
      PROGRAM TEST
C
C------------------------------------------------------------------
C     Program example for MPBNGC. Copyright by M.M.Mäkelä 2003.
C*******************************************************************
C
      INTEGER             N, MF, MG, MC, MP1, NIN, JMAX, LIWORK, LWORK
C
      PARAMETER          (N = 2, MF = 2, MG = 1, MC = 1, MP1 = MF+1,
     &                    NIN = 10, JMAX = 5,
     &                   LIWORK = 2*(MP1*(JMAX+1)+MC+N),
     &                   LWORK = MP1*(6*JMAX+10)+JMAX+5*MC+MG+MF+18+
     &                   N*(N+2*MP1*JMAX+2*MP1+2*MC+2*MG+2*MF+31)/2)
C
      INTEGER            LMAX, NITER, NFASG, NOUT, IPRINT, IERR,
     &                   IWORK(LIWORK), IUSER(1)
C
      DOUBLE PRECISION   RL, EPS, FEAS,
     &                   X(N), F(MF+MG), C(MC+1,N), B(MC+1), GAM(MP1),
     &                   BL(N), BU(N), WORK(LWORK), USER(1)
C
      EXTERNAL           FASG
C
      LMAX = 100
      NITER = 100
      NFASG = 100
C     Skip the heading of the data file
      READ (NIN,*)
      READ (NIN,*) (X(J), J=1,N)
      READ (NIN,*) (BL(J), J=1,N)
      READ (NIN,*) (BU(J), J=1,N)
      READ (NIN,*) ((C(I,J), J=1,N), B(I), I=1,MC)
      READ (NIN,*) (GAM(J), J=1,MP1)
      READ (NIN,*) RL, EPS, FEAS, IPRINT, NOUT
C-------------------------------------------------------------------
      CALL MPBNGC (N, X, MF, MG, F, FASG, MC, C, B, RL, LMAX,
     &             GAM, EPS, FEAS, JMAX, NITER, NFASG, NOUT,
     &             IPRINT, IERR, BL, BU, IWORK, LIWORK, WORK,
     &             LWORK, IUSER, USER)
C-------------------------------------------------------------------
      IF (IERR.GT.0) THEN
         WRITE (NOUT,*)
         WRITE (NOUT, 9999) 'MPBNGC terminated with IERR =', IERR
      END IF
      STOP
9999 FORMAT (1X,A,I3)
      END
C
C********************************************************************
C
      SUBROUTINE FASG (N, X, MM, F, G, IERR, IUSER, USER)
C
C********************************************************************
C
      INTEGER          N, MM, IERR, IUSER(*)
C
      DOUBLE PRECISION DIFF,
     &                 X(N), F(MM), G(N,MM), USER(*)
C
      INTRINSIC        DMAX1
C
C------------------
C     Rosenbrock
C------------------
      F(1) = 100*(X(2)-X(1)**2)**2+(1-X(1))**2
      G(1,1) = -400*X(1)*(X(2)-X(1)**2)-2*(1-X(1))
      G(2,1) = 200*(X(2)-X(1)**2)
C------------------
C     Crescent
C------------------
      F(2) = DMAX1(X(1)**2+(X(2)-1)**2+X(2)-1,
     &           -(X(1)**2)-(X(2)-1)**2+X(2)+1)
      DIFF = X(1)**2+(X(2)-1)**2+X(2)-1+(X(1)**2)+(X(2)-1)**2-X(2)-1
      IF (DIFF.GE.0.0D+00) THEN
         G(1,2) = 2*X(1)
         G(2,2) = 2*X(2)-1
      ELSE
         G(1,2) = -2*X(1)
         G(2,2) = -2*X(2)+3
      END IF
C------------------
C     Constraint
C------------------
      F(3) = (X(1)-1)**2+(X(2)-1)**2-1
      G(1,3) = 2*X(1)-2
      G(2,3) = 2*X(2)-2
      RETURN
      END
C
C********************************************************************
```

5.3. Program Data.

```
 Program Data for MPBNGC Example
   1.0D+00     0.0D+00
   0.0D+00     0.0D+00
   1.0D+00     1.0D+00
   1.0D+00     1.0D+00     1.0D+00
   0.3D+00     0.6D+00     0.0D+00
   0.1D-01     1.0D-05     1.0D-09     4     6
```

5.4. Program Results.

The output of MPBNGC is the following.

```
 Iter:  0 Nfun:  1   f1(x) = 100.0000       Eps = .4449887E-02
```

```
                    f2(x) = 1.000000
                     x(1) = 1.000000
                     x(2) = 0.0000000E+00
Iter:  1 Nfun:  2   f1(x) = 97.35182       Eps =0.1056127E-01
                    f2(x) = 0.9866899
                     x(1) = 0.9955501
                     x(2) = 0.4449887E-02
Iter:  2 Nfun:  3   f1(x) = 93.57143       Eps = 0.6191521E-01
                    f2(x) = 0.9674177
                     x(1) = 0.9884699
                     x(2) = 0.9750147E-02
Iter:  3 Nfun:  4   f1(x) = 70.62354       Eps = 0.4562671
                    f2(x) = 0.8432974
                     x(1) = 0.9458080
                     x(2) = 0.5419204E-01
Iter:  4 Nfun:  5   f1(x) = 70.62354       Eps = 0.3678280
                    f2(x) = 0.8432974
                     x(1) = 0.9458080
                     x(2) = 0.5419204E-01
Iter:  5 Nfun:  6   f1(x) = 9.432154       Eps = 0.1501864
                    f2(x) = 0.3698702
                     x(1) = 0.7474285
                     x(2) = 0.2545715
Iter:  6 Nfun:  7   f1(x) = 0.9172208       Eps = 0.7608890E-01
                    f2(x) = 0.3308772
                     x(1) = 0.5669369
                     x(2) = 0.2359963
Iter:  7 Nfun:  8   f1(x) = 0.9005946       Eps = 0.2598784E-01
                    f2(x) = 0.2393255
                     x(1) = 0.5066403
                     x(2) = 0.1756171
Iter:  8 Nfun:  9   f1(x) = 0.8817410       Eps = 0.8816035E-02
                    f2(x) = 0.2117380
                     x(1) = 0.4859587
                     x(2) = 0.1575746
Iter:  9 Nfun: 10   f1(x) = 0.8738081       Eps = 0.2961100E-02
                    f2(x) = 0.2027801
                     x(1) = 0.4788525
                     x(2) = 0.1516973
Iter: 10 Nfun: 11   f1(x) = 0.8709535       Eps = 0.9922325E-03
                    f2(x) = 0.1998081
                     x(1) = 0.4764441
                     x(2) = 0.1497434
Iter: 11 Nfun: 12   f1(x) = 0.8699735       Eps = 0.3326982E-03
                    f2(x) = 0.1988147
                     x(1) = 0.4756331
                     x(2) = 0.1490898
```

```
Iter:  12 Nfun:  13    f1(x) = 0.8696422        Eps = 0.1115930E-03
                       f2(x) = 0.1984819
                        x(1) = 0.4753607
                        x(2) = 0.1488707
Iter:  13 Nfun:  14    f1(x) = 0.8695307        Eps = 0.3743479E-04
                       f2(x) = 0.1983703
                        x(1) = 0.4752693
                        x(2) = 0.1487973
Iter:  14 Nfun:  15    f1(x) = 0.8694933        Eps = 0.1255835E-04
                       f2(x) = 0.1983329
                        x(1) = 0.4752386
                        x(2) = 0.1487726
Iter:  15 Nfun:  16    f1(x) = 0.8694808        Eps = 0.4213313E-05
                       f2(x) = 0.1983203
                        x(1) = 0.4752283
                        x(2) = 0.1487644
```

Note, that the previous version of MPBNGC utilizing QPDF4 as a quadratic solver terminated after 25 iteration and 26 function evaluation with accuracy difficulties (`IERR = 7`). The final solution was the point $x^{25} = (0.4752386, 0.1487726)$ and the objective function values were $f(x^{25}) = (0.8694933, 0.1983329)$ with the accuracy parameter value `Eps = 0.1522595E-04`.

6. Numerical Experiments

In order to show the reliability of the new version of MPBNGC, some numerical experiments with 10 nonsmooth unconstrained single criteria test problems described in [Haarala et al., 2003] will now be reported. The test problems can be formulated with any number of variables and they have constructed either by chaining and extending standard nonsmooth small scale problems or by nonsmoothing large scale smooth problems.

We compared the performance of the new version 2.0 of MPBNGC using PLQDF1 with the previous version 1.34 utilizing QPDF4 as a quadratic solver. The experiments were performed in the SGI Origin 2000/128 supercomputer (MIPS R12000, 600 Mflop/s/processor) of CSC.

The parameters had the following standard values: RL= 0.05, LMAX= 200, JMAX= 100 and EPS= 10^{-5}. In order not to restrict the solution process via limiting the function or subgradient evaluations the parameters NITER and NFASG were chosen to be large enough. For five convex problems the standard value 0.0 for GAM was used, while for nonconvex problems the value was 0.5.

The used dimensions of the problems were 10, 100 and 1 000. For the largest problems $n = 1\ 000$ the CPU time used were restricted to be less than half an hour. Our numerical results are summarized in Tables 1–3.

The following abbreviations will be used:

Ndif	–	number of problems with different results
Niter	–	the average number of iterations
Nfun	–	the average number of function calls
Nfail	–	the number of failures
Nlim	–	the number of time limits
CPU	–	the average CPU time in seconds.

The average results with 10 variables are presented in Table 1. The methods performed in quite a similar way; there where only two problems, where they differed. The version 2.0 used a little bit less iterations and function calls, but the older version 1.34 was a little bit faster.

Version	Ndif	Niter	Nfun	Nfail	Nlim	CPU
MPB 1.34	2	50.0	57.5	0	0	8.39E-03
MPB 2.00	2	48.1	50.8	0	0	8.71E-03

Table 1. Average results for $n = 10$.

The results obtained with 100 variables can be seen in Table 2. Now there were more differences in the performances, namely six. The new version was better in all the criteria measured: it used less iterations, function calls and CPU time than the old version. The differences were also clearer, especially in terms of CPU time.

Version	Ndif	Niter	Nfun	Nfail	Nlim	CPU
MPB 1.34	6	676.6	704.3	1	0	7.87
MPB 2.00	6	572.3	592.7	0	0	4.91

Table 2. Average results for $n = 100$.

Finally, the results with the large scale problems (1 000 variables) can be found in Table 3. The same trends as before seem to continue. There were differences in the performances in eight cases from ten. Again, the new version was better in all the criteria measured and the differences were now remarkable. One explanation for this is a test example, where the old version needed about ten times more iterations and function calls and 30 times more CPU time than the new one. Note, that the methods were terminated to the limit (half an hour) of CPU time by four times. These results are not included in the average numbers. However, in every case, the new version reached better solution and was faster to calculate more iterations in this time limit.

Version	Ndif	Niter	Nfun	Nfail	Nlim	CPU
MPB 1.34	8	219.5	291.1	1	4	33.43
MPB 2.00	8	45.2	48.4	0	4	1.26

Table 3. Average results for $n = 1000$.

7. Concluding Remarks

We have considered a new implementation version 2.0 of MPBNGC Fortran subroutine for nonlinearly constrained nonconvex multiobjective optimization. In the new version of MPBNGC the quadratic programming subproblem is solved by PLQDF1 subroutine, which is an realization of the dual range space quadratic programming method for minimax approximation with linear constraints implemented by Lukšan. The code PLQDF1 replaces QPDF4 subroutine realizing the dual active-set quadratic programming algorithm implemented by Kiwiel.

The numerical experiments show the reliability of the new version. We can conclude that it is more efficient and give more accurate results than the previous version. In the forthcoming versions of MPBNGC the main attention is devoted to numerically problematic equality constraints, both in linear and nonlinear cases.

Acknowledgements

We would like to thank PhD. Ladislav Lukšan for providing the quadratic solver PLQDF1 and PhLic. Marjo Haarala for numerical experiments.

This research was financially supported by TEKES, the National Technology Agency of Finland.

References

1. M. S. Bazaraa and C. M. Shetty, "Nonlinear Programming, Theory and Algorithms," John Wiley & Sons, New York, 1979.
2. F. H. Clarke, "Optimization and Nonsmooth Analysis," John Wiley & Sons, New York, 1983.
3. J. Haslinger and P. Neittaanmäki, "Finite Element Approximation for Optimal Shape, Material and Topology Design," John Wiley & Sons, Chichester, 1996.
4. K. C. Kiwiel, *A Descent Method for Nonsmooth Convex Multiobjective Minimization*, Large Scale Systems **8(2)** (1985(a)), 119–129.
5. K. C. Kiwiel, "Methods of Descent for Nondifferentiable Optimization," Lecture Notes in Mathematics 1133, Springer-Verlag, Berlin, Heidelberg, 1985(b).
6. K. C. Kiwiel, *A Method for Solving Certain Quadratic Programming Problems Arising in Nonsmooth Optimization*, IMA Journal of Numerical Analysis **6** (1986), 137–152.
7. K. C. Kiwiel, *Proximity Control in Bundle Methods for Convex Nondifferentiable Optimization.*, Mathematical Programming **46** (1990), 105–122.
8. K. C. Kiwiel, "User's Guide for NOA 4.0: A Fortran Package for Nonconvex Nondifferentiable Optimization," (Systems Research Institute, Polish Academy of Sciences, 1991.
9. C. Lemaréchal, *An Extension of Davidon Methods to Non Differentiable Problems*, Mathematical Programming Study **3** (1975), 95–109.

10. C. Lemaréchal, *Nondifferentiable Optimization*, in "Optimization, (Eds. G. L. Nemhauser, A. H. G. Rinnooy Kan and M. J. Todd)," North-Holland, Amsterdam, 1989, 529–572.
11. C. Lemaréchal and M. Bancora Imbert, "Le Module M1FC1," Technical Report, Institut de Recherche d'informatique et d'Automatique, Le Chesnay, 1985.
12. L. Lukšan, *Dual Method for Solving a Special Problem of Quadratic Programming as a Subproblem at Linearly Constrained Nonlinear Minimax Approximation*, Kybernetika **20** (1984), 445–457.
13. M. M. Mäkelä, "Issues of Implementing a Fortran Subroutine Package NSOLIB for Nonsmooth Optimization," University of Jyväskylä,, Department of Mathematics, Laboratory of Scientific Computing, Report 5/93, 1993.
14. M. M. Mäkelä, *Survey of Bundle Methods for Nonsmooth Optimization*, Optimization Methods and Software **17(1)** (2002), 1–29.
15. M. M. Mäkelä and P. Neittaanmäki, "Nonsmooth Optimization: Analysis and Algorithms with Applications to Optimal Control," World Scientific Publishing Co., Singapore, 1992.
16. K. Miettinen, "Nonlinear Multiobjective Optimization," Kluwer Academic Publishers, Boston, 1999.
17. K. Miettinen and M. M. Mäkelä, *Interactive Bundle-based Method for Nondifferentiable Multiobjective Optimization: NIMBUS*, Optimization **34** (1995), 231–246.
18. K. Miettinen and M. M. Mäkelä, *Interactive Multiobjective Optimization System WWW-NIMBUS on the Internet*, Computers & Operations Research **27** (2000), 709–723.
19. K. Miettinen and M. M. Mäkelä, "Synchronous Scalarizing Functions within the Interactive NIMBUS Method for Multiobjective Optimization," Reports of the Department of Mathematical Information Technology, No. B9/2002, University of Jyväskylä, 2002.
20. J. J. Moreau, P. D. Panagiotopoulos and Strang G. (Eds.), "Topics in Nonsmooth Mechanics," Birkhäuser, Basel, 1988.
21. J. Outrata, J. Zowe and H. Schramm, "Bundle Trust Methods: Fortran Codes for Nondifferentiable Optimization," User's Guide, DFG Report No. 269, 1991.
22. J. Outrata, M. Kočvara and J. Zowe, "Nonsmooth Approach to Optimization Problems with Equilibrium Constraints. Theory, Applications and Numerical Results," Kluwer Academic Publishers, Dordrecht, 1998.
23. R. T. Rockafellar, "Convex Analysis," Princeton University Press, Princeton, New Jersey, 1970.
24. H. Schramm and J. Zowe, *A Version of the Bundle Idea for Minimizing a Nonsmooth Functions: Conceptual Idea, Convergence Analysis, Numerical Results*, SIAM Journal on Optimization **2** (1992), 121–152.
25. S. Wang, *Algorithms for Multiobjective and Nonsmooth Optimization*, in "Methods of Operations Research, (Eds. P. Kleinschmidt, F. J. Radermacher, W. Schweitzer, H. Wildermann)," Athenäum Verlag, Frankfurt am Main, 1989, 131–142.