To appear in *Optimization Methods & Software*
Vol. 00, No. 00, Month 20XX, 1–16

Algorithmic Differentiation for Piecewise Smooth Functions: A Case Study for Robust Optimization

Sabrina Fiege[a*], Andrea Walther[a], Kshitij Kulshreshtha[a], and Andreas Griewank[b]

[a] *Department of Mathematics, Paderborn University, Paderborn, Germany*
[b] *School of Mathematical Science and Information Technology, Yachaytech, Urcuqui, Ecuador*

(*Received 00 Month 20XX; final version received 00 Month 20XX*)

This paper presents a minimization method for Lipschitz continuous, piecewise smooth objective functions based on algorithmic differentiation (AD). We assume that all nondifferentiabilities are caused by abs(), min(), and max(). The optimization method generates successively piecewise linearizations in *abs-normal form* and solves these local subproblems by exploiting the resulting kink structure. Both, the generation of the abs-normal form and the exploitation of the kink structure is possible due to extensions of standard AD tools. This work presents corresponding drivers for the AD tool ADOL-C which are imbedded in the nonsmooth solver LiPsMin. Finally, minimax problems from robust optimization are considered. Numerical results and a comparison of LiPsMin with other nonsmooth optimization methods are discussed.

Keywords: Piecewise linearization, Algorithmic differentiation, Nonsmooth optimization, Robust optimization

AMS Subject Classification: 90C26; 90C30; 90C47

1. Motivation

Nonsmoothness is a commonly occurring characteristic of optimization problems, e.g., problems from robust optimization. In that field minimax problems of the form

$$\min_{x\in X} f(x) \quad \text{where} \quad f(x) \equiv \max_{1\le i\le N}\{f_i(x),\ f_i \in C^1(X)\}$$

are typical, where N is the number of continuously differentiable functions f_i, $i = 1, ..., N$, so-called scenarios. Among other formulations, minimax-regret problems are considered where $f_i(x) \equiv \bar{f}_i(x) - \bar{f}_i^*$ with $\bar{f}_i(x)$ continuously differentiable and $\bar{f}_i^* \equiv \min_{x\in X} \bar{f}_i(x)$. Therewith, the worst-case regret of finitely many scenarios is minimized.

The objective functions of minimax-regret problems are Lipschitz continuous, piecewise smooth and all occurring nondifferentiabilities are caused by the maximum and minimum function and the absolute value function, respectively. Consequently, all nondifferentiabilities can be expressed in terms of the absolute value function. Hence, they are piecewise smooth (PS) functions in the sense of Scholtes [14, Chap. 4].

*Corresponding author. Email: sfiege@math.upb.de

There are still only few practical methods available for the minimization of Lipschitzian PS functions $f : \mathbb{R}^n \mapsto \mathbb{R}$. A common approach is to hope that the nondifferentiabilities do not affect standard algorithms designed for smooth problems too much. In fact as demonstrated by Lewis and Overton variants of the classical BFGS method do amazingly well and can be complemented by stochastic gradient sampling, see [10]. Another class of widely used methods are the bundle methods, see, e.g., [9]. However, the practical behavior of these algorithms at times seems to be rather erratic. Because of these observations, one may anticipate an improvement by gaining additional information about the structure of the nondifferentiabilities.

Our goal is the development of an optimization method that minimizes Lipschitz continuous, PS objective functions by successively generating piecewise linear models and to efficiently solve these local models by exploiting the polyhedral structure representing the nondifferentiabilities. This method is described in Sec. 3 and was motivated by a search trajectory proposed in the book of Hirriart-Urruty and Lemaréchal [8, Chap. 8].

Using an adapted version of the algorithmic differentiation tool ADOL-C, we are now able to provide all required information such as the directionally active gradients and the abs-normal form for our minimization method. Therewith, the extended version of ADOL-C including the resulting drivers is an essential component of the optimization method and is presented in Sec. 2.

In Sec. 4, test problems originating from robust optimization are introduced and numerical results are presented. Finally, a conclusion and an outlook are given in Sec. 5.

2. New Functionalities and Drivers of ADOL-C

The objective functions $f : \mathbb{R}^n \to \mathbb{R}$ considered in this work belong to the class of PS functions that are defined in [14, Chap. 4] and can be represented as

$$f(x) \in \{f_\sigma(x) :\ \sigma \in \mathcal{E} \subset \{-1, 0, 1\}^s\} \quad \text{at} \quad x \in \mathbb{R}^n,$$

where the finitely many selection functions f_σ are continuously differentiable on neighborhoods of points where they are active, i.e., coincide with f. We will assume that all f_σ with $\sigma \in \mathcal{E}$ are essential in that their coincidence sets $\{f(x) = f_\sigma(x)\}$ are the closures of their interiors. For PS functions of this form, Clarke's subdifferential $\partial f(x)$ can be given as the convex hull of the limiting subdifferential $\partial^L f(x)$, i.e.,

$$\partial f(x) \equiv \operatorname{conv}(\partial^L f(x)) \quad \text{with} \quad \partial^L f(x) \equiv \{\nabla f_\sigma(x) :\ f_\sigma(x) = f(x)\},$$

where the elements of $\partial^L f(x)$ are called limiting gradients of f at x. For the computation of elements of the limiting subdifferential $\partial^L f(x)$ by the AD tool ADOL-C, see [15], requires an adapted evaluation procedure for the considered objective functions which will be introduced in the following section.

Furthermore, it will be explained how the piecewise linearization is generated. This piecewise linearization is the key ingredient of the local model. It was introduced in [3], and can be written in its abs-normal form, as explained in [7]. The abs-normal form allows us to exploit the structure of the argument space caused by the nondifferentiabilities. In the remainder of this section, the abs-normal form and the structure exploitation will be explained in more detail and the resulting drivers are introduced.

Adapted Evaluation Procedure

We will consider only objective functions $f : \mathbb{R}^n \mapsto \mathbb{R}$ that can be evaluated by a sequence of elementary functions which are either Lipschitz continuously differentiable in an open domain $D \subset \mathbb{R}^n$ or the absolute value function. Using the reformulations

$$\min(v,u) = (v+u-\text{abs}(v-u))/2 \quad \text{and}$$
$$\max(v,u) = (v+u+\text{abs}(v-u))/2,$$

a quite large range of piecewise differentiable and locally Lipschitz continuous functions is covered by this assumption. Conceptually combining consecutive smooth elemental functions into larger smooth elemental functions, one obtains the reduced evaluation procedure shown in Tab. 1, where all evaluations of the absolute value function can be clearly identified and exploited. The number of absolute values occurring during the function evaluation will be denote by $s \in \mathbb{N}$. The arguments z_i of the absolute value functions cause the switches in the corresponding derivative values and therefore, $z = (z_i) \in \mathbb{R}^s$ is called switching vector. Furthermore, the switching vector defines the signature vector $\sigma = (\sigma_i) = (\text{sign}(z_i)) \in \mathbb{R}^s$.

Table 1. Adapted evaluation procedure

v_{i-n}	$=$	x_i	$i = 1 \ldots n$
z_i	$=$	$\psi_i(v_j)_{j \prec i}$	$i = 1 \ldots s$
σ_i	$=$	$\text{sign}(z_i)$	
v_i	$=$	$\sigma_i z_i = \text{abs}(z_i)$	
y	$=$	$\psi_{s+1}(v_j)_{j \prec s+1}$	

The user does not have to identify the switching variables z_i, $i = 1, ..., s$, let alone specify their dependencies or the connecting super elementary functions ψ_i, $i = 1, ..., s+1$. All this is handled automatically by the AD-tool ADOL-C [15], which provides numerical values of the ψ_i and their derivatives as needed. In the original evaluation procedure of ADOL-C, where functions are assumed to be compositions of exclusively Lipschitz continuously differentiable elemental functions, the absolute value function causes possibly an error message depending on the evaluation point. In contrast to that, the currently available, adapted ADOL-C 2.6.3 additionally offers the explicit handling of the signature vector avoiding these error messages.

Piecewise Linearization

To obtain a piecewise linearization of the target function f, one has to construct for each smooth elemental function a tangent approximation as described in [4, Chap. 3]. For a given argument x and a direction Δx, we will use the elemental linearizations from standard AD theory, extended by the following rule for abs() as proposed in [3].

$$\begin{aligned}
\Delta v_i &= \Delta v_j \pm \Delta v_k && \text{for } v_i = v_j \pm v_k, \\
\Delta v_i &= v_j * \Delta v_k + v_k * \Delta v_j && \text{for } v_i = v_j * v_k, \\
\Delta v_i &= \varphi'(v_j)_{j \prec i} * \Delta(v_j)_{j \prec i} && \text{for } v_i = \varphi_i(v_j)_{j \prec i} \neq \text{abs}(v_j), \\
\Delta v_i &= \text{abs}(v_j + \Delta v_j) - v_i && \text{for } v_i = \text{abs}(v_j). \qquad (1)
\end{aligned}$$

This extension of the usual AD approach can be used to compute the increment $\Delta f(x; \Delta x)$ and therefore the piecewise linearization (PL)

$$f_{PL,x}(\Delta x) = f(x) + \Delta f(x; \Delta x) \tag{2}$$

of the original PS function f at a given point x with the incremental argument Δx.

Example 2.1 *We consider the piecewise smooth and nonconvex function*

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x_1, x_2) = (x_2^2 - (x_1)_+)_+ \quad \textit{with} \quad y_+ \equiv \max(0, y). \tag{3}$$

Its piecewise linearization $f_{PL,\bar{x}}$ evaluated at a base point $\bar{x}$ and with the argument Δx is given by

$$f_{PL,\bar{x}}(\Delta x) = \frac{1}{2}\left(z_2 + |z_2|\right) \tag{4}$$

with $z_1 = \bar{x}_1 + \Delta x_1$ and $z_2 = \bar{x}_2^2 + 2\bar{x}_2 \Delta x_2 - \frac{1}{2}\left(z_1 + |z_1|\right)$.

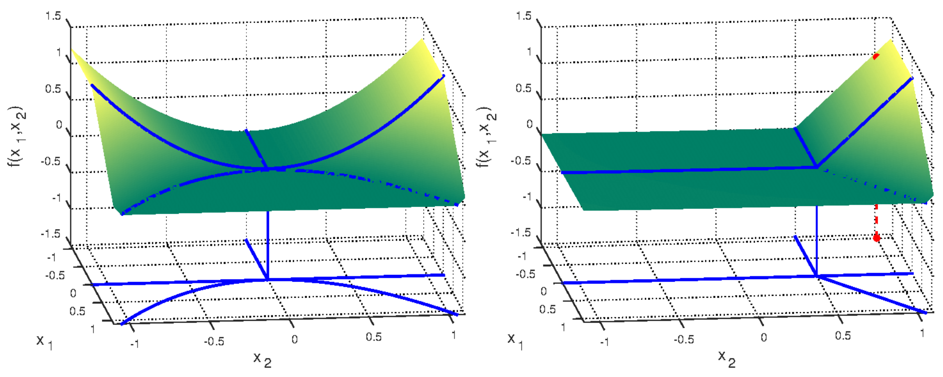

Figure 1. Left: The piecewise smooth function (3), Right: The corresponding piecewise linearization (4) evaluated in $\bar{x} = (-1, 1)$

To generate the corresponding piecewise linearization with the AD-tool ADOL-C [15], one has to perform the following steps: First, an internal representation of the function, which gets a unique identifier `tag`, has to be generated. Therefore, the corresponding mode of ADOL-C for the differentiation of PS functions has to be enable by using

```
enableMinMaxUsingAbs().
```

Afterwards, all dependent variables as well as all required intermediate values have to be computed for given independent variables between the commands `trace_on()` and `trace_off()`. Subsequently, the call

```
zos_pl_forward(tag,1,n,1,x,y,z);
```

with the argument x as the point at which the piecewise linearization is evaluated, provides the switching vector z according to Tab. 1 and the function value $y = f(x)$. Note, that the switching vector z, also yields the signature vector σ. Here, the abbreviation zos stand for zero-order-scalar signaling that only a function evaluation and no derivative calculation is performed and the information is propagated forward through the evaluation procedure.

One can extract the number of absolute value function evaluations using the call

```
s=get_num_switches(tag);
```

where s is defined in Tab. 1. This yields also the dimension of the vectors z and σ. Subsequently, one can use the driver

```
fos_pl_forward(tag, 1, n, x, deltax, y, deltay, z, deltaz);
```

to actually compute the increment $\Delta y = \Delta f(x;\Delta x)$ using the piecewise linear approximation (1). As output variables, one has also the switching vector z and its piecewise linearization Δz. The abbreviation fos stands for first-order-scalar mode. Additionally, the first-order-vector mode is also available. Hence, for every piecewise smooth function given as C or C++ code it is now tractable to generate a piecewise linear model at a given argument x.

Structure of Decomposed Domain for Piecewise Linearizations

To compute a limiting gradient $\nabla f_\sigma(x)$ of an essential selection function f_σ of a PL at a nondifferential point x, it is necessary to analyze the polyhedral decomposition induced by the nondifferentiabilities. This structure of the PL is given by its decomposition into polyhedra P_σ. For any PL, it follows by continuity that P_σ must be open but possibly empty if σ is *definite* in that all its components are nonzero. Generally we have for any nonempty P_σ

$$\dim(P_\sigma) \;\geq\; n + \|\sigma\|_1 - s = n - s + \sum_{i=1}^{s} |\sigma_i| \; .$$

When equality holds we call the signature σ *nondegenerate* and otherwise *critical.* In particular degenerate situations there may be some critical σ that are nevertheless *open* in that P_σ is open. We certainly have by definition of $\sigma = \sigma(x)$ for the closure of P_σ

$$\bar{P}_\sigma \;\subset\; \{x \in \mathbb{R}^n : f_{PL,x}(x) = f_\sigma(x)\},$$

where identity must hold in the convex case. In the nonconvex case f_σ may coincidentally be active, i.e., coincide with $f_{PL,x}$ at points in other polyhedra $P_{\tilde{\sigma}}$. Particulary, f_σ is essentially active in the sense of Scholtes [14, Chap. 4] at all points in $\bar{P}_\sigma$ provided σ is open. Whether or not it is essentially active somewhere outside of $\bar{P}_\sigma$ is irrelevant. To conform with the general concepts of PS functions we may restrict f_σ to some open neighborhood of $\bar{P}_\sigma$ such that it cannot be essentially active outside P_σ. The corresponding signature vectors are called *essential* and are given by

$$\mathcal{E} = \{\sigma \in \{-1,0,1\}^s : \emptyset \neq P_\sigma \text{ open}\}.$$

Furthermore, the signature vectors have a partial order of the form

$$\sigma \preceq \tilde{\sigma} \quad :\Longleftrightarrow \quad \sigma_i^2 \leq \tilde{\sigma}_i \sigma_i \quad \text{for} \quad 1 \leq i \leq s$$

which gives more insight of the polyhedral decomposition. Generally, we will describe the decomposition primarily in terms of the signature vectors σ, as illustrated in Exam. 2.2. More information about the polyhedral structure and its description by signature vectors can be found in [6].

Example 2.2 *The domain of the piecewise linearization* (4) *is decomposed by two absolute value functions into four open polyhedra with corresponding nonzero signatures* $\sigma = (\pm 1, \pm 1)$ *as can be seen in Fig. 2.*

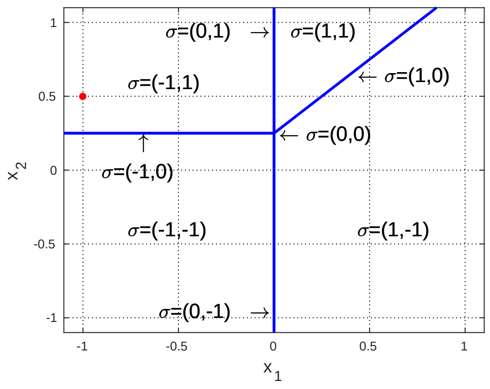

Figure 2. The piecewise linearization (4) evaluated in $\bar{x} = (-1, 0.5)$

At this point, it is necessary to distinguish between two different evaluations of signature vectors. On the one hand, there is the evaluation of the signature vector $\sigma(x)$ depending on the current point x which was introduced in Tab. 1 and which is given by

$$\sigma = (\sigma_i(x))_{i=1,...,s} \equiv (\text{sign}(z_i(x)))_{i=1,...,s} \in \mathbb{R}^s.$$

On the other hand, we are interested in the signature vectors of polyhedra corresponding to essential selection functions of the PL which are active at nondifferential points x. Such a signature vector $\sigma(x; \Delta x)$ is defined by

$$\sigma(x; \Delta x) = (\sigma_i(x; \Delta x))_{i=1,...,s} \equiv (\text{firstsign}(z_i(x); \nabla z_i^\top(x; \Delta x)E))_{i=1,...,s} \qquad (5)$$

where $E \in \mathbb{R}^{n \times n}$ is a regular matrix and firstsign(z;$\nabla z^\top(x; \Delta x)E$)) returns for each component σ_i, $i = 1, ..., s$, the sign of the first nonvanishing entry of the vector $(z_i(x); \nabla z_i^\top(x; \Delta x)E) \in \mathbb{R}^{n+1}$. For our application we choose

$$E = [\Delta x, e_1, ..., e_{j^*-1}, e_{j^*+1}, e_n] \quad \text{with} \quad j^* = \text{argmax}_{j=1,...,n} |\Delta x_j|,$$

where e_i, $i = 1, ..., n$, are the unit vectors.

In [3] more information about the firstsign function can be found. It is indicated that one might prefer to start with less than n directions and thus, to increase the number of directions iteratively until the obtained signature vector is definite for the first time or n is reached. Because of these considerations the evaluation of $\sigma(x; \Delta x)$ is implemented as follows:

```
 short firstsign(int p, double *z, double* deltaz) {
     int i=0, tmp;
     tmp=((*z)>0.0)?1.0:(((*z)<0.0)?-1.0:0.0);
     while(i<p && tmp==0.0) {
          tmp=(deltaz[i]>0.0)?1.0:((deltaz[i]<0.0)?-1.0:0.0);
          i++;
     }
     return tmp;
}
```

The routine gets $z \equiv z(x)$ and $\nabla z \equiv \nabla z(x; E)$ which is denoted *deltaz* in the implementation and depends already on $E \in \mathbb{R}^{n\times p}$ as input variables and returns $\sigma(x; \Delta x)$. To obtain $\sigma(x, \Delta x)$ an extension of the driver fos_pl_forward() of the form

fos_pl_sig_forward(tag, 1, n, x, deltax, s, sigx, y, deltay, z, deltaz, sigxdx);

was included in ADOL-C where sigx= $\sigma(x)$ is the signature vector at the current point x and one obtains sigxdx= $\sigma(x, \Delta x)$ as an additional return value.

Directionally Active Gradient

To evaluate a directionally active gradient at a certain point x a direction d is required that indicates where the active domain of the selection function is located. This is especially important if x is a boundary point of the active domain such that several essential selection functions are active at x. Detailed information on these gradients can be found in [3]. Formally, a *directionally active gradient* g is given by

$$g \equiv g(x; d) \in \partial^L f(x) \quad \text{such that} \quad f'(x; d) = g^\top d \tag{6}$$

and $g(x; d)$ equals the gradient $\nabla f_\sigma(x)$ of a locally differentiable selection function f_σ that coincides with f on a set, whose tangent cone at x contains d and has a nonempty interior.

If the function evaluation is coded in C or C++ and the internal representation of ADOL-C is available, the driver

directional_active_gradient(tag, n, x, d, g, sigxd);

can be used to compute $g(x; d)$ at a given point x and a given direction d. Similar to the firstsign function, this driver evaluates the directionally active gradient by iteratively adding directions until for the first time the corresponding signature vector $\sigma(x; d)$ is definite. Since the number of necessary directions is in most cases very small, the total number of directions in the implementation is limited as can be seen in the following sketch in pseudo code.

```
int directional_active_gradient(tag, n, x, d, g, sigxd){
 keep = 1; by = 1; k = 1; done = 0; j = 0;
 s=get_num_switches(tag);
 E = [d];
 max_entry = max_i fabs(d[i]);
 max_dk= argmax_i fabs(d[i]);

 while((k<limitd) && (done == 0)){
     fov_pl_forward(tag,1,n,k,x,E,y,deltay,z,deltaz,sigxd);

     sum = 0;
     for(i=0;i<s;i++)
         sum += fabs(sigxd[i]);

     if (sum == s){
         zos_pl_forward(tag,1,n,keep,x,&y,z);
         fos_pl_sig_reverse(tag,1,n,s,sigxd, &by ,g);
         done = 1;
     }else{
         if(j==max_dk)
           j++;
         E=[E e_j];
         j++; k++;
     }
 }
```

```
    if (done == 0){
        \\ Compute g(x;d) with full E
        ...
    }
}
```

Abs-normal Form

To exploit the nondifferentiabilities of the PL in the minimization algorithm described in Sec. 3, we rewrite the PL in its abs-normal form. In [3] it was shown that any piecewise linear function $y = f_{PL}(\Delta x)$ with

$$f_{PL} : \mathbb{R}^n \to \mathbb{R}^m$$

can be expressed using the argument Δx and the resulting switching vector $z \in \mathbb{R}^s$ in the *abs-normal* form given by

$$\begin{bmatrix} z \\ y \end{bmatrix} = \begin{bmatrix} c_z \\ c_y \end{bmatrix} + \begin{bmatrix} Z & L \\ Y & J \end{bmatrix} \begin{bmatrix} \Delta x \\ |z| \end{bmatrix}, \tag{7}$$

where $c_z \in \mathbb{R}^s$, $c_y \in \mathbb{R}^m$, $Z \in \mathbb{R}^{s\times n}$, $L \in \mathbb{R}^{s\times s}$, $Y \in \mathbb{R}^{m\times n}$ and $J \in \mathbb{R}^{m\times s}$. The matrices Y and J are row vectors in this optimization context, since we consider functions with $m = 1$. Correspondingly, c_y is only a real number instead of a vector. The matrix L is strictly lower triangular, i.e., each z_i is assumed to be an affine function of absolute values $|z_j|$ with $j < i$ and the input values x_k for $1 \le k \le n$. The structural nilpotency degree of L, i.e., the smallest number $\mu \le s$ such that

$$L^\mu = 0, \tag{8}$$

is called switching depth of f in the given representation. Defining the signature matrix

$$\Sigma \equiv \Sigma(\Delta x) \equiv \operatorname{diag}(\sigma(\Delta x)) \ \in \{-1, 0, 1\}^{s\times s}$$

for the switching variables of the piecewise linearization as defined in Eq. (5), one obtains for a fixed $\sigma \in \{-1, 0, 1\}^s$ and $|z| \equiv \Sigma\, z$ for z from the first equation of Eq. (7) that

$$(I - L\Sigma)z = c_z + Z\Delta x \quad \text{and} \quad z \ = \ (I - L\Sigma)^{-1}(c_z + Z\Delta x).$$

Notice that due to the strict triangularity of $L\,\Sigma$ the inverse $(I - L\Sigma)^{-1}$ is well defined and polynomial in the entries of $L\,\Sigma$. Substituting this expression into the last equation of Eq. (7), it follows for the function value that

$$f_\sigma(\Delta x) \equiv \gamma_\sigma + g_\sigma^\top \Delta x \tag{9}$$

with

$$\gamma_\sigma = c_y + J\Sigma(I - L\Sigma)^{-1}c_z \quad \text{and} \quad g_\sigma^\top = Y + J\Sigma(I - L\Sigma)^{-1}Z.$$

That is, the gradient evaluation for the piecewise linearization reduces to solving a linear system with a triangular matrix. This will be exploited for a cheap gradient calculation in the inner loop of the optimization algorithm presented in the next section.

ADOL-C provides the driver

```
abs_normal(tag, m, n, s, x, sigma, y, z, cz, cy, Y, J, Z, L);
```

to compute the components of the abs-normal form for a given PS function f and a given base point x. It is important to note, that the parts c_z, c_y, Y, J, Z, and L only depend on the considered PS function f and the base point x in which the PL is evaluated. To compute the abs-normal form efficiently, a further driver of the form

```
fos_pl_reverse(tag, m, n, s, i, res);
```

is required which applies the adapted handling of the absolute value function. The routine returns res=[Z_i L_i] for $i = 0, ..., s-1$ and res=[Y_i J_i] for $i = s, ..., m+s-1$ where Z_i, L_i, Y_i and J_i denote the *i-th* row of the corresponding matrix. Now one can compute the components cz and cy by solving the two equations of the abs-normal form for these components. Since the abs-normal form is evaluated in the base point x, it yields additionally that with $\Delta x = 0$, one obtains

$$c_z = z - L|z| \qquad \text{and} \qquad c_y = y - J|z|.$$

Hence, all components required by the minimization method LiPsMin can be provided by the adapted version of ADOL-C.

3. Minimization of Lipschitz continuous, piecewise smooth functions

In the following, the minimization method LiPsMin for nonsmooth optimization problems of the form

$$\min_{x \in \mathbb{R}^n} f(x)$$

is described. The objective function $f : \mathbb{R}^n \to \mathbb{R}$ is assumed to be a piecewise smooth function as defined in [14] and all nondifferentiabilities are coded in terms of the maximum, minimum and absolute value function. The optimization method consists of an outer loop that successively generates piecewise linear local models that are complemented by an quadratic penalty term. These local quadratic models are minimized by an inner loop which exploits the polyhedral structure caused by the nondifferentiabilities as described in the previous section. A detailed description of the method including convergence theory is given in [2].

Minimization algorithm for piecewise smooth functions

As sketched already in [3], we propose the following algorithm to minimize Lipschitzian piecewise smooth functions:

Algorithm 3.1 (LiPsMin)

```
**LiPsMin**($x$, $q^0$, $q^{lb}$, $\kappa$) // *Precondition: $x \in \mathbb{R}^n$, $\kappa > 0$, $q^0, q^{lb} > 0$ sufficiently large*
$x^0 = x$
for $k = 0, 1, 2, \ldots$
```

1. Generate a PL model $f_{PL,x^k}(.)$ at the current iterate x^k.
2. Use PLMin(x^k, Δx^k, q^k) to solve the overestimated local probem

$$\Delta x^k = \underset{\Delta x \in \mathbb{R}^n}{\arg\min} f_{PL,x^k}(\Delta x) + \frac{1}{2}(1+\kappa)q^k\|\Delta x\|^2. \tag{10}$$

3. Set $x^{k+1} = x^k + \Delta x^k$ if $f(x^k + \Delta x^k) < f(x^k)$ and $x^{k+1} = x^k$ else.
4. Compute

$$\hat{q}^{k+1} \equiv \hat{q}(x^k, \Delta x^k) \equiv \frac{2|f(x^{k+1}) - f_{PL,x^k}(\Delta x^k)|}{\|\Delta x^k\|^2}$$

and set $q^{k+1} = \max\{\hat{q}^{k+1}, \mu\, q^k + (1-\mu)\,\hat{q}^{k+1}, q^{lb}\}$ with $\mu \in [0,1]$.

As can be seen, one main ingredient of the approach is the successive piecewise linearization and the solution of the overestimated local model (10) by Algo. 3.2. The overestimation is necessary to ensure lower boundedness of the model and to obtain the required convergence behavior. So far, there is no termination criterion given such that the algorithm generates an infinite sequence of iterates $\{x^k\}$ that can be exploited in the convergence analysis. A reasonable termination criterion is to check if $\|\Delta x^k\| = 0$ after step 2, since from this it follows that the piecewise smooth function f is Clarke stationary in the current iterate x_k, as proven in [2].

Minimization Algorithm for Piecewise Linear Functions

PLMin solves the local model (10) by exploiting the structure induced by the nondifferentiabilities. Therefore, the polyhedral structure of the decomposition was analyzed above and in more detail in [5, 6]. Since we consider the k-th iteration of Algo. 3.1 subsequently, we use x, Δx, and q instead of x^k, Δx^k and q^k for simplicity. The essential difference between PLMin and the *true descent algorithm* introduced in [6, Algo. 4] is the solution of a sequence of special quadratic subproblems along a path of essential polyhedra instead of computing a critical step multiplier for a given direction which turned out to be numerically unstable.

Assume that $j-1$ quadratic subproblems were already solved with the solutions Δx_l, $l = 0, ..., j-1$. In order to solve the j-th subproblem these previous solutions have to be included such that the relationship between the current essential polyhedron P_{σ^j} and and the base point x is maintained. Hence, one obtains the following subproblem

$$\delta x_j = \underset{\delta x \in \mathbb{R}^n}{\arg\min} f_{\sigma^j}\left(\Delta x_j + \delta x\right) + \frac{\breve{q}}{2}\left\|\Delta x_j + \delta x\right\|^2, \tag{11}$$
$$\text{s.t.} \quad z_i(x) + \nabla z_i(x)^\top\left(\Delta x_j + \delta x\right) \begin{cases} \leq 0 & \text{if } \sigma_i^j < 0 \\ \geq 0 & \text{if } \sigma_i^j > 0 \end{cases} \qquad \text{for} \quad i = 1, \ldots, s\ ,$$

where $\Delta x_j \equiv \sum_{l=0}^{j-1} \delta x_l$, $\breve{q} \equiv (1+\kappa)q$, $z_i(x)$ is the i-th component of the switching variable of $z(x)$ and $\nabla z_i(x)$ the corresponding gradient. All required components are available from the abs-normal form. Hence, using the drivers presented in the previous section, all required information can be computed. Minimizing the quadratic objective (11) subject to the given constraints by any QP solver will signal whether P_{σ^j} is empty or not. Due to the proximal term added to the piecewise linear local model the objective function is positive definite quadratic on P_{σ^j}.

The above approach can be summarized as follows, where the base point x and the quadratic coefficient q serve as input variables, the increment Δx as the output parameter.

Algorithm 3.2 (PLMin)

```
**PLMin**($x$, $\Delta x$, $\breve{q}$) // Precondition: $x, \Delta x \in \mathbb{R}^n$, $\breve{q} \geq 0$
Set $\Delta x_0 = 0$. Identify $\sigma^0 = \sigma(x)$.
For $j = 0, 1, 2, ...$
    1.  Determine solution $\delta x_j$ of local QP (11) on current polyhedron $P_{\sigma^j}$.
    2.  Update $\Delta x_{j+1} = \Delta x_j + \delta x_j$.
    3.  Compute direction $d$.
    4.  If $\|d\| = 0$: STOP.
    5.  Identify new polyhedron $P_{\sigma^{j+1}}$ using direction $d$.
return $\Delta x = \Delta x_{j+1}$
```

The main remaining challenge is to decide how to change the signature at a minimizer Δx_{j+1} in order to move to a neighboring polyhedron where the function value decreases in step 5 of Algo. 3.2. In other words we have to find a descent direction d at Δx_{j+1} and a signature σ^{j+1} such that $P_{\sigma^{j+1}}$ contains $\Delta x_{j+1} + \tau d$ for small positive τ.

In [2] it was proposed to build a bundle of the limiting subdifferential of the PL $f_{PL,x}$ at the current iterate Δx_{j+1}. Initially it contains the gradient g_{σ^j} of the current selection function f_{σ^j}. The direction d is defined as $d = -\mathrm{short}(qx, G)$ with

$$\mathrm{short}(h, G) = \arg\min \left\{ \|d\| \,\middle|\, d = \sum_{j=1}^{m} \lambda_j g_j - h,\ g_j \in G,\ \lambda_j \geq 0,\ \sum_{j=1}^{m} \lambda_j = 1 \right\}.$$

Afterwards, the bundle G gets augmented by further directionally active gradients $g(x; d)$ as defined in Eq. (6) corresponding to neighboring polyhedra.

Algo. 3.2 converges to a stationary point Δx^* after finitely many steps, since the argument space is divided only into finitely many polyhedra, the local model $\hat{f}_x$ is bounded below and the function value is decreased each time we switch from one polyhedron to another polyhedron.

Furthermore, if $f : \mathbb{R}^n \to \mathbb{R}$ is a piecewise smooth function as described in Sec. 2 which has a bounded level set $\mathcal{N}_0 = \{x \in \mathbb{R}^n \mid f(x) \leq f(x^0)\}$ with x^0 the starting point of the generated sequence of iterates $\{x^k\}_{k\in\mathbb{N}}$, then a cluster point x^* of the infinite sequence $\{x^k\}_{k\in\mathbb{N}}$ generated by Algo. 3.1 exists. All cluster points of the infinite sequence $\{x^k\}_{k\in\mathbb{N}}$ are Clarke stationary as proven in [2].

4. Examples from Robust Optimization

In this section, the performance of the introduced nonsmooth optimization algorithm LiPsMin is analyzed and compared with other state-of-the-art nonsmooth optimization software. Therefore, test problems originated from robust optimization, so-called minimax problems, are considered. Minimax problems minimize the possible loss of a worst case scenario. A common loss function is the absolute regret, i.e., the gap between the function and its global minimizer. These problems are known as minimax-regret problems.

The test problems considered in this section consist of two minimax-regret problems and three general minimax problems. Further information about these test problems can be found in [11].

- **minimax-regret problem 1** (based on QL)

$$f : \mathbb{R}^2 \to \mathbb{R},\ f(x) = \max_{1\le i\le 3}\{f_i(x)\}$$

with $f_i(x) \equiv \bar{f}_i(x) - \bar{f}_i^*$, $i = 1, 2, 3$, $\bar{f}_1^* = 0$, $\bar{f}_2^* = -385$, $\bar{f}_3^* = -65$, and

$$\begin{aligned}
\bar{f}_1(x) &= x_1^2 + x_2^2,\\
\bar{f}_2(x) &= \bar{f}_1(x) + 10(-4x_1 - x_2 + 4),\\
\bar{f}_3(x) &= \bar{f}_1(x) + 10(-x_1 - 2x_2 + 6)
\end{aligned}$$

Starting point: $x^0 = (-1, 5)$, Optimal value: $f(x^*) = 106.25$

- **minimax-regret problem 2** (based on Rosen-Suzuki)

$$f : \mathbb{R}^4 \to \mathbb{R},\ f(x) = \max_{1\le i\le 4}\{f_i(x)\}$$

with $f_i(x) \equiv \bar{f}_i(x) - \bar{f}_i^*$, $i = 1, ..., 4$, $\bar{f}_1^* = -79,875$, $\bar{f}_2^* = -88.407$, $\bar{f}_3^* = -114.706$, $\bar{f}_4^* = -69.230$, and

$$\begin{aligned}
\bar{f}_1(x) &= x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4,\\
\bar{f}_2(x) &= \bar{f}_1(x) + 10(x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_1 - x_2 + x_3 - x_4 - 8),\\
\bar{f}_3(x) &= \bar{f}_1(x) + 10(x_1^2 + 2x_2^2 + x_3^2 + 2x_4^2 - x_1 - x_4 - 10),\\
\bar{f}_4(x) &= \bar{f}_1(x) + 10(2x_1^2 + x_2^2 + x_3^2 + 2x_1 - x_2 - x_4 - 5)
\end{aligned}$$

Starting point: $x^0 = 0$, Optimal value: $f(x^*) = 37.220432$

- **minimax problem 1** (Davidon 2)

$$f : \mathbb{R}^4 \to \mathbb{R},\ f(x) = \max_{1\le i\le 20}\{f_i(x)\}$$

with $t_i = 0.2i$ and $f_i(x) = (x_1 + x_2 t_i - \exp(t_i))^2 + (x_3 + x_4\sin(t_i) - \cos(t_i))^2$

Starting point: $x^0 = (25, 5, -5, -1)$, Optimal value: $f(x^*) = 115.70644$

- **minimax problem 2** (maxquad)

$$\begin{aligned}
f : \mathbb{R}^{10} \to \mathbb{R},\ f(x) &= \max_{1\le i\le 5} x^\top A^i x - x^\top b^i\\
\text{with}\quad A^i_{kj} = A^i_{jk} &= e^{j/k}\cos(jk)\sin(i), \quad i < k,\\
A^i_{jj} &= \frac{j}{10}\left|\sin(i)\right| + \sum_{k\ne j}\left|A^i_{jk}\right|,\\
b^i_j &= e^{j/i}\sin(ij)
\end{aligned}$$

Starting point: $x^0 = 0$, Optimal value: $f(x^*) = -0.8414083$

- **minimax problem 3** (maxq)

$$f : \mathbb{R}^n \to \mathbb{R},\ f(x) = \max_{1 \le i \le n} x_i^2$$

Starting point: $x_i^0 = i$ for all $i = 1, ..., n/2$ and $x_i^0 = -i$ for all $i = n/2 + 1, ..., n$, Optimal value: $f(x^*) = 0$

Next, the presented algorithm LiPsMin is compared with the nonsmooth optimization routines MPBNGC, i.e., a proximal bundle method described in [12], and the quasi-Newton type method HANSO described in [10].

The idea of the bundle method MPBNGC is to approximate the subdifferential of the objective function at the current iterate by collecting subgradients of previous iterates and storing them into a bundle. Thus, more information about the local behavior of the function is available. For a detailed description see [12, 13].

The quasi-Newton type method HANSO 2.2 implements a BFGS method and combines the BFGS method with an inexact line search and if required a gradient sampling approach described in [1]. The later two software packages assume that the user provides not only a routine to evaluate the function but also a routine that evaluates an analytical subgradient. Which in turn means that the user has to know the actual subgradient or has to compute the expression by hand which is an error prone process.

As stopping criteria of routines LiPsMin and PLMin we used $\epsilon = 1e - 4$ and the maximal iteration number $maxIter = 1000$. In the implementation of Algo. 3.1 we chose the parameters $\mu = 0.9$ and $q^{lb} = q^0 = 0.1$. For the bundle method MPBNGC we choose the following parameter settings: The maximal bundle size equals the dimension n. Further stopping criteria are the number of iterations NITER= 10000, the number of function and gradient evaluations NFASG= 10000 and the final accuracy `EPS`= $1e - 8$. For HANSO 2.2 we choose the default values except $maxit = 10000$, i.e., we use the BFGS mode with line search but without gradient sampling if not otherwise specified.

		f^*	$\#f$	$\#\nabla f$	Iter
minimax-regret prob. 1	LiPsMin	106.25	18	34	17
	HANSO	106.25	36	36	20
	MPBNGC	106.25	44	44	43
minimax-regret prob. 2	LiPsMin	37.220431	63	186	62
	HANSO	37.220430	199	199	73
	MPBNGC	37.220429	93	93	85
minimax problem 1	LiPsMin	115.706	48	893	47
	LiPsMin + Sparsity	115.706	48	423	47
	HANSO	115.706	443	443	186
	MPBNGC	115.706	104	104	87
minimax problem 2	LiPsMin	-0.841429	33	128	32
	HANSO	0	32	32	1
	HANSO + GS	-0.841402	2368	2368	1 + 3 GS
	MPBNGC	-0.841408	40	40	39

Table 2. Results for the minimax-regret problems 1 and 2, and the minimax problems 1 and 2

The results of the two minimax-regret problems and of the minimax problems 1 and 2 are illustrated in Tab. 2. The table contains all the results of the test problems generated by the three different optimization routines mentioned. The columns of the table give the final function value f^*, the number of function evaluations $\#f$, the number of gradient evaluations $\#\nabla f$ and the number of iterations. For LiPsMin, $\#\nabla f$ counts the number of reverse sweeps needed to compute the abs-normal form (ANF), i.e., $\#\nabla f = s \cdot \#ANF$. Here, one has to note, that the representation of the function may have a considerable impact on the reverse sweeps needed to compute the abs-normal form. Using a naive evaluation, the matrices Z and L are dense such that indeed $s \cdot \#ANF$ reverse sweeps are needed. If the function representation is coded such that the switching depth ν is minimized the matrices Z and L are quite often very sparse. Then, the exploitation of sparsity may yield a dramatic reduction of reverse sweeps as illustrated in Tab. 2 by the minimax problem 1. The vast majority of optimization runs detected successfully minimal solutions. The only exception is a result obtained by HANSO. In this case, the routine terminated at a non minimal point. However, the minimal solution was also detected by HANSO after enabling gradient sampling. Nevertheless, the number of function and gradient evaluations increased significantly. In general, the optimization tools need a comparable number of function and gradient evaluations.

	n	f^*	$\#f$	$\#\nabla f$	Iter
LiPsMin	10	8.73e-10	35	306	34
	20	4.02e-9	37	684	36
	50	2.29e-9	62	2989	62
	100	8.09e-9	124	12177	123
LiPsMin + Sparsity	10	8.73e-10	35	170	34
	20	4.02e-9	37	216	36
	50	2.29e-9	62	427	62
	100	8.09e-9	124	984	123
HANSO	10	4.10e-9	160	160	92
	20	8.00e-9	414	414	224
	50	1.37e-8	965	965	480
	100	2.37e-8	2003	2003	949
MPBNGC	10	3.45e-9	126	126	101
	20	2.60e-9	244	244	222
	50	3.79e-9	577	577	549
	100	4.46e-9	1118	1118	1083

Table 3. Results for minimax problem 3

In Tab. 3 the results of the third minimax problem are summarized. We show this problem in a extra table because it is scalable in its dimension n and thus, we can study the behavior of the optimization tools with an increasing number of optimization parameters. First of all, all optimization runs succeeded and as one would expect, the number of function and gradient evaluations grow with an increasing number of optimization parameters. However, it becomes apparent, that, although the number of iterations performerd by LiPsMin are less than for the other software tools, the number of gradient evaluations needed by LiPsMin increase faster. This is especially the fact, when a possible sparsity of the matrices Z and L is ignored. By using compression techniques for these highly sparse matrices, the amount of gradient evaluations can be decreased considerably.

Fig. 3 gives a first hint of the convergence rate of LiPsMin. The figure shows how the function value $f(x^k)$ of the k-th iteration decreases during an optimization run for $n = 10$ and $n = 100$. Considering these results LiPsMin seems to converge quadratic under certain conditions which will be analyzed in greater detail in future work.

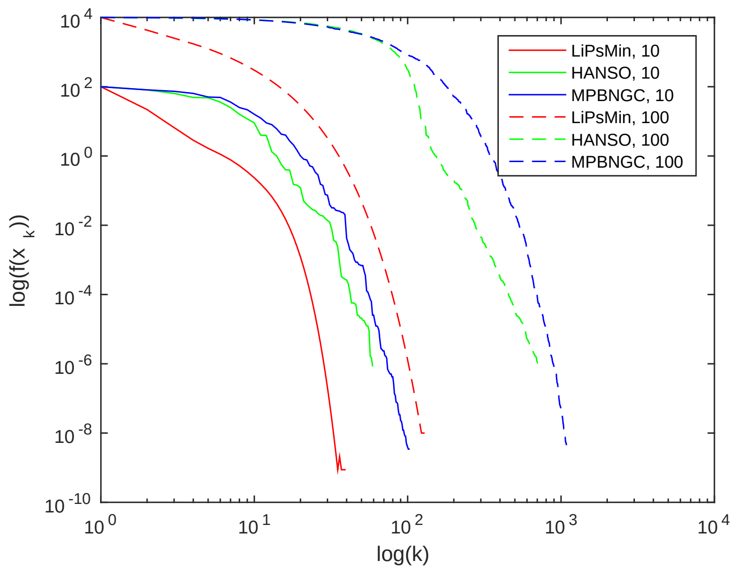

Figure 3. Convergence behavior of LiPsMin, HANSO and MPBNGC for minimax problem 3 with $n = 10$ and $n = 100$

In conclusion, we can say that LiPsMin compares well with state-of-the-art optimization tools as HANSO and MPBNGC, requiring always the least number of iterations.

5. Conclusion and Outlook

In [2] we introduced the algorithm LiPsMin which minimizes Lipschitz continuous, piecewise smooth functions by successive piecewise linearization. That paper focused on a detailed theoretical description of the algorithms PLMin and LiPsMin including several description of required components such as the abs-normal form and directionally active gradients, and an attractive convergence result for LiPsMin.

In this work, LiPsMin is examined explicitly how the components required by LiPsMin can be computed via an adapted version of ADOL-C, as discussed in Sec. 2. The key ingredient of the adapted ADOL-C version was certainly the integration of the rule for the piecewise linearization of the absolute value function. It allowed us to establish correspondingly adapted zero- and first-order drivers for both scalar and vector forward mode, as well as a first order reverse driver. Therewith, the function value and piecewise linearizations of the function in a certain point can be computed. Furthermore, two easy-to-use drivers were included: One driver evaluates the abs-normal form and the other driver returns a directionally active gradient.

In Sec. 4, minimax problems originating from robust optimization theory were considered. The performance results of LiPsMin confirmed well our expectations and it compared well with the state-of-the-art optimization software HANSO and MPBNGC.

Computing the new direction in Algo. 3.2 with the bundle-based approach can be computational expensive. That is why it should be an important aspect of future work to incorporate the optimality conditions of [5] such that the descent direction can be computed more efficiently. Hence, it would be beneficial to gain more information about the polyhedral decomposition of the domain, such as convexity properties of the function. The additional information can be used to identify the subsequent polyhedron more efficiently which is especially relevant when the considered function is high dimensional.

Acknowledgement

The authors gratefully acknowledge the support of the project WA 1607/12-1 within the SPP 1962 by the German Research Foundation.

References

[1] J. Burke, A. Lewis, and M. Overton, *A robust gradient sampling algorithm for nonsmooth nonconvex optimization.*, SIAM J. Optim. 15 (2005), pp. 751–779.

[2] S. Fiege, A. Walther, and A. Griewank, *An algorithm for nonsmooth optimization by successive piecewise linearization*. submitted, available on www.optimization-online.org.

[3] A. Griewank, *On stable piecewise linearization and generalized algorithmic differentiation*, Optimization Methods and Software 28 (2013), pp. 1139–1178.

[4] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*, SIAM, 2008.

[5] A. Griewank and A. Walther, *First and second order optimality conditions for piecewise smooth objective functions*, Optimization Methods and Software 31 (2016), pp. 904 – 930.

[6] A. Griewank, A. Walther, S. Fiege, and T. Bosse, *On Lipschitz optimization based on gray-box piecewise linearization*, Mathematical Programming, Series A (2015), pp. 1–33.

[7] A. Griewank, J.U. Bernt, M. Radons, and T. Streubel, *Solving piecewise linear systems in abs-normal form*, Linear Algebra and its Applications 471 (2015), pp. 500 – 530.

[8] J.B. Hiriart-Urruty and C. Lemaréchal, *Convex Analysis and Minimization Algorithms I*, Springer, 1993.

[9] N. Karmitsa and M. Mäkelä, *Limited memory bundle method for large bound constrained nonsmooth optimization: convergence analysis.*, Optimization Methods and Software 25 (2010), pp. 895–916.

[10] A. Lewis and M. Overton, *Nonsmooth optimization via quasi-Newton methods.*, Mathematical Programming 141 (2013), pp. 135–163.

[11] L. Lukšan and J. Vlček, *Test problems for nonsmooth unconstrained and linearly constrained optimization*, Technical Report 798, Institute of Computer Science, Academy of Sciences of the Czech Republic, 2000.

[12] M. Mäkelä and P. Neittaanmäki, *Nonsmooth Optimization: Analysis and Algorithms with Applications to Optimal Control*, World Scientific Publishing Co., 1992.

[13] M.M. Mäkelä, *Multiobjective proximal bundle method for nonconvex nonsmooth optimization: Fortran subroutine MPBNGC 2.0*, Reports of the Department of Mathematical Information Technology, Series B, Scientific computing No. B 13/2003, University of Jyväskylä, 2003.

[14] S. Scholtes, *Introduction to Piecewise Differentiable Functions*, Springer, 2012.

[15] A. Walther and A. Griewank, *Combinatorial Scientific Computing*, chap. Getting Started with ADOL-C, Chapman-Hall CRC Computational Science, 2012, pp. 181–202.