OBOE User Guide
Draft Edition

Jean-Philippe Vial[1]
Nidhi Sawhney[2]

June 21, 2006

[1]vial@hec.unige.ch
[2]sawhney@hec.unige.ch

Abstract

OBOE(Oracle Based Optimization Engine) is a software package developed for solving convex non-differentiable minimization problems over a convex set. In this document we give a brief overview of the underlying ACCPM theory used in OBOE, and how to use OBOE for solving Non-Differentiable Optimization (NDO) problems. We also present some benchmark problems solved using OBOE.

Contents

Chapter 1

Introduction to ACCPM

ACCPM is a class of computational methods used to solve the convex problems of finding a point in a convex set or minimizing a convex function over a convex set. The methods we shall describe use polyhedral relaxations of convex sets and functions. They are based on the property that a convex set can be described as the intersection of the half-spaces that contain it. A finite subset of such half-spaces defines a polyhedral relaxation of the set. We name *polyhedral relaxation method*, a method that is based on this property. Since the half-spaces are bounded by separating hyperplanes that cut the space into two half-spaces, the terminology *cutting plane* to denote the separating hyperplane, and by extension for the method is often used.

The basic assumption is that there exists an *oracle* that produces separation hyperplanes. Namely, given a convex set $C \subset R^n$ and a point $y \in R^n$, the oracle returns the information $y \in C$, or produces a vector $a \in R^n$ and a scalar $c \leq 0$, which separate y from C

$$a^T(y' - y) \leq c, \forall y' \in C.$$

The first problem that arise in connection with an oracle for C is the feasibility problem

$$\text{find} \quad y \in C. \tag{1.1}$$

The set C is often given in functional form

$$C = \{y \mid g(y) \leq 0\}.$$

In that case the oracle for C is a numerical procedure to compute $g(y)$ and an element $\xi \in \partial g(y)$ in the subdifferential set of g at y. This oracle works as follows: if $g(y) \leq 0$, the oracle confirms that $y \in C$; else, the oracle produces the cut

$$\xi^T(y' - y) + g(y) \leq 0, \forall y' \in C.$$

The pure feasibility problem may be converted into a simple optimization problem by adding an objective function

$$\min\{b^T y \mid y \in C\}.$$

The simple minimization problem allows to treat the more conventional problem of minimizing a convex function f by simple embedding into the epigraph space of f

$$\min\{z \mid f(y) - z \leq 0\}.$$

The more general case deals with the minimization of a function that is the sum of two convex functions, one smooth and the other not. Since the sum of the two components is nonsmooth, the solution method belongs to the realm of nondifferentiable optimization. However, standard methods do not exploit the second order information that can be extracted from the smooth component. We propose an adaptation of polyhedral relaxation methods to exploit this information.

The more general problem —which from now on is our canonical problem— takes the form

$$\min\{f(y) = f_1(y) + f_2(y) \mid y \in Y_1 \cap Y_2\}, \tag{1.2}$$

where f_1 and f_2 are convex functions on R^n and $Y_i \subset R^n$, $i = 1, 2$, The function f_1 and the set Y_1 are revealed by a first order oracle. In a general setting f_2 is taken to be a convex twice continious differentiable function while Y_2 is a 0-level set of a convex twice continuously differenciable function $g(y)$. In applications, we further assume that f_2 is self-concordant and Y_2 is endowed with a self-concordant barrier.

We shall consider two main possibilities for Y_2. Either $Y_2 = R^n$, or

$$Y_2 = \{y \mid g_i(y) \leq 0,\ i = 1, \ldots, r\}.$$

The more relevant cases g_i are linear or convex quadratic. The function are written as

$$g_i(y) = \frac{1}{2} y^T P_i y + p_i^T y + d_i \leq 0.$$

In general, Y_2 includes simple box constraints $\beta_l \leq y \leq \beta_u$ or a ball constraint $||y - y^c|| \leq R$.
Note: OBOE currently only supports the above box and ball constraints.

Finally, the nonsmooth function f_1 often is the positively weighted sum of p nonsmooth functions

$$f_1(y) = \sum_{i=1}^{p} \pi_i f_{1i}(y).$$

This property can be exploited in the solution method.

The canonical problem (1.2) can be written in format similar to the simple minimization problem

$$\begin{aligned}
\min \quad & \pi^T z + \zeta \\
& f_{1j}(y) - z_j \leq 0,\ j = 1, \ldots, p, && (1.3)\\
& f_2(y) - \zeta \leq 0, && (1.4)\\
& g_i(y) \qquad \leq 0,\ i = 1, \ldots, r, && (1.5)\\
& y \in Y_1. && (1.6)
\end{aligned}$$

The new problem is an embedding of the original problem in an $R^n \times R^p$ dimensional space. If $f_2(y)$ is a linear function given by $b^T y$, we replace ζ by $b^T y$ and do away with equation 1.4.

Problem (1.2) often arises in connection with a dualization (or partial dualization) scheme. In particular, we have in mind Lagrangian relaxation schemes which generate dual problems of the canonical type. To be consistent with this important segment

of the literature, we shall consider that Problem 1.2 is the dual problem of some (unspecified) primal problem. The variables y, z and ζ will be named "dual". Later, we will work with the duals of problems written in the dual variables. We will call these problems "primal".

In some applications the variables can be additionally constrained to lie in an affine set $\{y \mid D^T y = d\}$.

1.1 Polyhedral relaxation schemes

Suppose the oracle has been querried at $(y^1, \ldots, y^k)$. The oracle has returned a set of feasability and/or optimality cuts. The resulting inequalities together with the linear constraints in $g_i(y) \leq 0$ are collected in the inequalities

$$A^T y - E^T z \leq c$$

In that definition, E is a simple matrix that is constructed as follows. If the nonsmooth objective is not disaggregated ($p = 1$), then E would be a row vector of 0 and 1. The 1 indicates that the z variable is present in the cut, meaning that the cut is an *optimality cut* associated with f_1. In contrast, a 0 indicates that the cut is a *feasibility cut* associated with Y_1. If the nonsmooth objective is disaggregated into p components, E is a $p \times m$ matrix of the form

$$E = \begin{pmatrix} 1\ldots1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1\ldots1 & \cdots & 0 & \vdots & \ddots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1\ldots1 & 0 & \cdots & 0 \end{pmatrix}.$$

Each row in that matrix correspond to a variable z_j and each column to a cut. A 1 in row j and column k indicates that the cut k is an optimality cut emanating from $f_{1j}(y)$. If column k is a null vector, then cut k is a feasibility cut. When the oracle produces an optimality cut for an element f_{1j}, it is the rule that it produces an element for each component of f_1. Consequently, as soon as there is an optimality cut, the matrix E has full row rank and its rows are orthogonal.

We intersect the set with Y_2 and we add the constraint that the objective achieves at least the best recorded value. If the best value is achieved at $\bar{y}$ the constraint takes the form

$$\pi^T z + \zeta \leq \bar{\theta} = f_1(\bar{y}) + f_2(\bar{y}),$$

where $\pi^T z$ is a surrogate for the nonsmooth objective and ζ the smooth objective. The set we described is named the *set of localization*. It is formally written

$$\mathcal{L}_{\bar{\theta}} = \{(y, z, \zeta) \mid A^T y - E^T z \leq c,\ \pi^T z + \zeta \leq \bar{\theta},\ f_2(y) \leq \zeta,\ y \in Y_2\}. \tag{1.7}$$

Notice also that the last inequalities in the definition of the set of localization are nonlinear if f_2 is non-linear. As mentioned before for the special case f_2 is linear we do not need the surrogate ζ and directly use the linear funtion.

The basic step of a cutting plane method can be defined as follows.

1. Select a query point in the set of localization.

2. Send the query point to the oracle and retrieve the information.

3. Update the lower and upper bounds and the set of localization.

4. Test termination.

The set of localization may or may not include an objective cut. Methods differ in the selection of the query point, in the updating of the localization set, in the availability of of a lower bound and in the termination criterion.

1.2 Lower bound on the opimal value

We propose a general scheme to compute a lower bound for the optimal value. In this derivation we assume that the problem is formulated with box constraints $y_l \leq y \leq y_u$. The bounds can be well-defined and given explicitly, or they are conservative estimates on the location of optimal solutions. In the former case, the lower bound on the optimal value will be exact; in the latter, it will simply be an estimate.

The optimal value is

$$\theta^* = \min\{f_1(y) + f_2(y) \mid y_l \leq y \leq y_u,\ y \in Y_1 \cap Y_2\}. \tag{1.8}$$

We suppose that the epigraph of f_1 and the set Y_1 are approximated by the set of inequalities $A^T y - E^T z \leq c$. This means that for all feasible y

$$f_1(y) \geq \min\{\pi^T z \mid E^T z \geq A^T y - c\}.$$

By convexity, we also have

$$f_2(y) \geq f_2(\bar{y}) + (f_2'(\bar{y}))^T (y - \bar{y}),$$

where $\bar{y}$ is an arbitrary reference point. Since the inequalities are valid for all feasible y, we may write

$$\begin{aligned} \theta^* &\geq \min_{y_l \leq y \leq y_u} \left\{ f_2(\bar{y}) + (f_2'(\bar{y}))^T (y - \bar{y}) + \min_z \{\pi^T z \mid E^T z \geq A^T y - c\} \right\}, \\ &\geq f_2(\bar{y}) - (f_2'(\bar{y}))^T (\bar{y}) + \eta, \end{aligned}$$

where

$$\eta = \min_{y,z} \left\{ (f_2'(y))^T y + \pi^T z \mid A^T y - E^T z \leq c,\ y_l \leq y \leq y_u \right\}.$$

The dual of this minimization problem is

$$\begin{aligned} \max \quad & -c^T \xi - y_u^T \psi_u + y_l^T \psi_l \\ & A\xi + f_2'(\bar{y}) + \psi_u - \psi_l = 0, \\ & E\xi = \pi, \\ & \xi \geq 0, \psi_u \geq 0, \psi_l \geq 0. \end{aligned}$$

Suppose we are given a non zero vector $\xi \geq 0$ such that $E\xi = \pi$. We can easily find feasible values for ψ_l and ψ_u that maximize the dual objective. Indeed

$$-y_u^T \psi_u + y_l^T \psi_l = -(y_u - y_l)^T \psi_u + y_l^T (\psi_l - \psi_u) = -(y_u - y_l)^T \psi_u + y_l^T (A\xi + f_2'(\bar{y})).$$

Since $y_u - y_l \geq 0$, then ψ_u must be chosen as small as possible. But $\psi_u \geq 0$ and

$$\psi_u = \psi_l - (A\xi + f_2'(\bar{y})) \geq -(A\xi + f_2'(\bar{y})).$$

Thus, the values that maximize the objective are

$$\begin{aligned} \psi_u &= (A\xi + f_2'(\bar{y}))_- \\ \psi_l &= (A\xi + f_2'(\bar{y}))_+. \end{aligned}$$

We thus have the exact lower bound

$$\underline{\theta} = -c^T\xi - y_u^T r_- + y_l^T r_+, \tag{1.9}$$

where $r = A\xi + f_2'(\bar{y})$.

To implement (1.9) we need to give values for $\bar{y}$, ξ, y_l and y_u. The natural choices for $\bar{y}$ in the proximal analytic center method (to be discussed later) are either the proximal reference point or the last approximate analytic center. The latter choice potentially yields a value for $r = A\xi + f_2'(\bar{y})$ closer to zero. (We shall even discuss later a method to further decrease r.) The proximal analytic center cutting plane method also produces a vector $\xi > 0$ that approximately satisfies $E\xi \approx \pi$. Due to the special structure of E, we can always scale $\xi > 0$ to have $E\xi = \pi$. Finally, if $y_u > y_l$ are given explicitly, we obtain an exact lower bound.

If y_u and/or y_l and/or some of the components of the two vectors are not given explicitly, we must estimate them by making some assumption on the distance of $\bar{y}$ to the optimal set. To this end, we write

$$-y_u^T r_- + y_l^T r_+ = r^T\bar{y} - r_-^T(y_u - \bar{y}) - r_+^T(\bar{y} - y_l).$$

The numerical value of the bound is obtained by stating component-wise upper bounds on $0 \leq y_u - \bar{y}$ and $0 \leq \bar{y} - y_l$.

1.3 Proximal generalized analytic centers

We now propose an implementation of the generic cutting plane method based on analytic center.

1.3.1 Barriers for the localization set

We associate with constraints of the localization set a standard (weighted) logarithmic barrier.

$$F(\bar{s}) = \sum_{i=0}^{m} F_i(s_i) + F_s(\sigma) = -\sum_{i=0}^{mr} w_i \log s_i - \omega \log \sigma,$$

with $\bar{s} = (s_0, s, \sigma) > 0$ defined by

$$\begin{aligned} s_0 &= \bar{\theta} - (\pi^T z + \zeta) \\ s_i &= c_i - (A^T y - E^T z)_j, \quad i \in I = \{1, \ldots, m\}, \\ \sigma &= \zeta - f_2(y). \end{aligned}$$

We also assume that the set Y_2 is endowed with a self-concordant barrier $H(y)$. In most applications the set Y_2 is defined by simple constraints.

Ball constraint This constraint restricts the points to be in a ball of radius R centered at y^r. The barrier

$$H(y) = -\log(R^2 - ||y - y^r||^2)$$

is self-concordant with self-concordant parameter $\nu = 1$.

Box constraints The simple box constraints take the form $\beta_l \leq y \leq \beta_u$. The associated barrier is

$$H(y) = -\sum_{i=1}^{n}(\log(y - \beta_l)_i + \log(\beta_u - y)_i).$$

From a formal point of view, these constraints can be treated just as simple feasibility constraints that are introduced at the outset. In the implementation of the algorithm, the structure of these constraints is exploited to reduce the complexity of the linear algebra.

The barrier function is augmented with a proximal term to yield the augmented barrier

$$\frac{\rho}{2}(y - \bar{y})^T Q(y - \bar{y}) + F(\bar{s}) + H(y),$$

where Q is a positive definite matrix. In view of the original minimization problem, we compound the generalized augmented barrier with the objective $\pi^T z$ of the polyhedral approximation of the nonsmooth objective $f_1(y)$ and the objective ζ of the polyhedral approximation of the smooth objective $f_2(y)$. We get

$$G(y, z, \zeta) = \frac{\rho}{2}(y - \bar{y})^T Q(y - \bar{y}) + \pi^T z + \zeta + F(\bar{s}(y, z, \zeta)) + H(y). \tag{1.10}$$

In the interior point literature, problems in which the nonnegativity constraints apply to the slacks of linear inequalities are considered to be in the *dual format*. Barrier functions of the type $G(y, z, \zeta)$ are also referred to as *dual potentials*. We shall use this terminology, despite the confusing fact that our dual potential is applied to our primal problem (1.3).

The proximal generalized analytic center polyhedral method defines the query point for the nonsmooth oracle as the y component of the solution of

$$\begin{aligned}
\min \quad & \frac{\rho}{2}(y - \bar{y})^T Q(y - \bar{y}) + H(y) \\
& \qquad + \pi^T z + \zeta - \sum_{i=0}^{m} w_i \log s_i - \omega \log \sigma && (1.11)\\
& s_0 = \bar{\theta} - (\pi^T z + \zeta) > 0, && (1.12)\\
& s_i = c_i - (A^T y - E^T z)_i > 0, \quad i = 1, \ldots, m && (1.13)\\
& \sigma = \zeta - f_2(y) > 0. && (1.14)
\end{aligned}$$

We shall denote the objective $G(y, z, \zeta)$.

To make sure that the minimum exists, we introduce the assumption

Assumption 1 *The vector π is positive and the set of localization has a non-empty interior.*

Theorem 1 *Under assumption 1, the minimum exists and is unique.*

For the sake of simplicity, we shall later use the notation $u = (y, z, \zeta)$ and write $G(u)$ for $G(y, z, \zeta)$. Note that G is a barrier for the localization set. It is self-concordant, but it is not a ν-normal barrier.

To write the generalized analytic center problem in a more condensed format, we introduce the variable $u^T = (y^T, z^T, \zeta^T)$ and collect all the constraints into $h(u) \leq 0$. Some components of h are associated with the cutting planes approximating f_1 and Y_1; the other are just the g functions representing Y_2 and the nonlinear constraints. We find it convenient to formulate the constraints as

$$h(u) + \tilde{s} = 0,\ \tilde{s} = (s_0, s_1, \cdots, s_m, \sigma) \geq 0.$$

Finally, writing

$$K(u) = \frac{\rho}{2}(y - \bar{y})^T Q(y - \bar{y}) + \pi^T z + \zeta + H(y),$$

we formulate the generalized analytic center problem as

$$u^c = (y^c, z^c, \zeta^c) = \arg\min_{u,s}\{K(u) + F(\bar{s}) \mid h(u) + \bar{s} = 0,\ \bar{s} > 0\}. \tag{1.15}$$

1.3.2 First order optimality

The first order optimality conditions are

$$\begin{aligned}
\rho Q(y - \bar{y}) + H'(y) + \omega\sigma^{-1} f_2'(y) + Ax &= 0, && (1.16)\\
(1 + w_0 s_0^{-1})\pi - Ex &= 0, && (1.17)\\
(1 + w_0 s_0^{-1}) - \omega\sigma^{-1} &= 0, && (1.18)\\
s_0 + (\pi^T z + \zeta) - \theta &= 0, && (1.19)\\
s + A^T y - E^T z - c &= 0, && (1.20)\\
\sigma + f_2(y) - \zeta &= 0. && (1.21)
\end{aligned}$$

It is possible to interpret $x_i = w_i s_i^{-1}$ and $\xi = \omega\sigma^{-1}$ as "primal" variables. We then have the complementary condition $x_i s_i = w_i$, $i = 0, 1, \ldots m$, and $\xi\sigma = \omega$.

Recall that if f_2 is linear we replace ζ with the linear term $b^T y$ and there is no need for variable σ. We use eqaution 1.18 in equation 1.16 to get
$\rho Q(y - \bar{y}) + H'(y) + (1 + w_0 s_0^{-1})b + Ax = 0$

1.4 Solving the Newton system

The aim is to minimize $G(u) = K(u) + F(-h(u))$. The method of choice is Newton's method. Let us first briefly review the case of a feasible Newton method. The Newton direction is

$$du = -[G''(u)]^{-1} G'(u).$$

The variant of Newton's method for computing the proximal generalized analytic center consists in taking damped steps to preserve feasibility of y and z. The aim is to achieve a sufficient decrease of G, until the area of quadratic convergence is hit. From then on, the method takes full Newton steps, with no line-search. We recall that the sufficient condition for guaranteed quadratic convergence is

$$\langle [G''(u)]^{-1} G'(u), G'(u) \rangle = \langle -du, G'(u) \rangle < 1. \tag{1.22}$$

The left-hand side of the inequality is a proximity measure. When the proximity is below the unit threshold, then the point $u + du$ is feasible and quadratically closer to the generalized analytic center (smaller proximity measure). The stopping criterion is a threshold value $\eta < 1$ on the proximity. To enforce this proximity at $u + du$, it suffices that the proximity at u be less than $\sqrt{\eta}$.

The stopping criterion (1.22) does not imply that $G'(u) = 0$, but most likely $G'(u)$ will be close to zero and $G'(u + du)$ even closer.

1.5 Infeasible start

Problem (1.15) raises the issue of feasibility. In cutting plane schemes, the new constraints exclude the current iterate from the new localization set. There is no direct way to retrieve feasibility if the cuts are deep. We propose an infeasible start Newton method, which aims to achieve feasibility and optimality simultaneously.

Let us explicit the the first and second derivatives. (Recall that h is linear.)

$$\begin{aligned} G'(u) &= K'(u) - \frac{\partial h}{\partial u} F'(-h(u)), \\ G''(u) &= K''(u) + \frac{\partial h}{\partial u} F''(-h(u)) \frac{\partial h}{\partial u}^T. \end{aligned}$$

Using the intermediate slack variable $\bar{s} = (s_0, s, \sigma)$, we obtain for the first optimality conditions for (1.15)

$$\begin{aligned} K'(u) - \frac{\partial h}{\partial u} F'(\bar{s}) &= 0, \\ h(u) + \bar{s} &= 0. \end{aligned}$$

In the course of the optimization process, those equations are never satisfied. However, we assume that $\bar{s} > 0$, and we introduce the residual r and write

$$K'(u) - \frac{\partial h}{\partial u} F'(\bar{s}) = r_d, \tag{1.23}$$

$$h(u) + \bar{s} = r_p. \tag{1.24}$$

The Newton direction in the $(du, d\bar{s})$ space with $d\bar{s} = (ds_0, ds, d\sigma)$ is given by

$$K''(u)du \quad - \frac{\partial h}{\partial u} F''(\bar{s}) d\bar{s} = -r_d, \tag{1.25}$$

$$\frac{\partial h}{\partial u}^T du \qquad + d\bar{s} \qquad = -r_p. \tag{1.26}$$

1.6 General case: nonlinear smooth component

Prior to discussing ways of solving (1.25)-(1.26), we explicit the components in the equations (1.23) to (1.26). We have

$$K'(u) = \begin{pmatrix} \rho Q(y - \bar{y}) + H'(y) \\ \pi \\ 1 \end{pmatrix},$$

$$K''(u) = \begin{pmatrix} \rho Q + H''(y) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix},$$

and

$$\frac{\partial h}{\partial u} = \begin{pmatrix} 0 & A & f_2'(y) \\ \pi & -E & 0 \\ 1 & 0 & -1 \end{pmatrix}.$$

Finally we have

$$F'(\bar{s}) = -\bar{w}\bar{s}^{-1},$$

with $\bar{w} = (w_0, w, \omega)$

$$F''(s) = \bar{W}\bar{S}^{-2}.$$

We recall the first order optimality conditions

$$\begin{aligned}
\rho Q(y - \bar{y}) + \omega f_2'(y)\sigma^{-1} + H'(y) + AWs^{-1} &= 0, && (1.27)\\
-EWs^{-1} + \pi w_0 s_0^{-1} &= -\pi, && (1.28)\\
w_0 s_0^{-1} - \omega\sigma^{-1} &= -1, && (1.29)\\
\pi^T z + \zeta + s_0 &= \theta, && (1.30)\\
A^T y - E^T z + s &= c, && (1.31)\\
f_2(y) - \zeta + \sigma &= 0. && (1.32)
\end{aligned}$$

The system to be solved takes the form

$$\begin{aligned}
(\rho Q + \omega f_2''(y)\sigma^{-1} + H''(y))dy - AWS^{-2}ds - \omega f_2'(y)\sigma^{-2}d\sigma &= r_y, && (1.33)\\
EWS^{-2}ds - \pi w_0 s_0^{-2} ds_0 &= r_z, && (1.34)\\
-w_0 s_0^{-2} ds_0 + \omega\sigma^{-2}d\sigma &= r_\zeta, && (1.35)\\
\pi^T dz + d\zeta + ds_0 &= r_{s_0}, && (1.36)\\
A^T dy - E^T dz + ds &= r_s, && (1.37)\\
f_2'(y)dy - d\zeta + d\sigma &= r_\sigma, && (1.38)
\end{aligned}$$

with residuals defined by

$$\begin{aligned}
r_y &= -(\rho Q(y - \bar{y}) + \omega f_2'(y)\sigma^{-1} + H'(y) + AWs^{-1}),\\
r_z &= -(-EWs^{-1} + \pi w_0 s_0^{-1} + t\pi),\\
r_\zeta &= -(w_0 s_0^{-1} - \omega\sigma^{-1} + t),\\
r_{s_0} &= -(\pi^T z + \zeta + s_0 - \theta),\\
r_s &= -(A^T y - E^T z + s - c),\\
r_\sigma &= -(f_2(y) - \zeta + \sigma).
\end{aligned}$$

Let us write the matrix associated with the system (1.33)–(1.38). We have

$$N = \begin{pmatrix} \Delta & 0 & 0 & 0 & -A\Lambda & -\omega f_2'(y)\sigma^{-2} \\ 0 & 0 & 0 & -\pi w_0 s_0^{-2} & E\Lambda & 0 \\ 0 & 0 & 0 & -w_0 s_0^{-2} & 0 & \omega\sigma^{-2} \\ 0 & \pi^T & 1 & 1 & 0 & 0 \\ A^T & -E^T & 0 & 0 & I & 0 \\ f_2'(y)^T & 0 & -1 & 0 & 0 & 1 \end{pmatrix},$$

with

$$\Delta = \rho Q + \omega f_2''(y)\sigma^{-1} + H''(y)$$

and

$$\Lambda = WS^{-2}.$$

Denote

$$M = \Delta + A\Lambda A^T + \omega\sigma^{-2} f_2'(y) f_2'(y)^T.$$

After pivoting on the SE diagonal block of N, we obtain the equivalent system:

$$\begin{pmatrix} M & -A\Lambda E^T & -\omega\sigma^{-2} f_2'(y) \\ -E\Lambda A^T & E\Lambda E^T + w_0 s_0^{-2}\pi\pi^T & w_0 s_0^{-2}\pi \\ -\omega\sigma^{-2} f_2'(y)^T & w_0 s_0^{-2}\pi^T & w_0 s_0^{-2} + \omega\sigma^{-2} \end{pmatrix} \begin{pmatrix} dy \\ dz \\ d\zeta \end{pmatrix}$$
$$= \begin{pmatrix} r_y + A\Lambda r_s + \omega\sigma^{-2} r_\sigma f_2'(y) \\ r_z - E\Lambda r_s + w_0 s_0^{-2} r_{s_0}\pi \\ r_\zeta + w_0 s_0^{-2} r_{s_0} - \omega\sigma^{-2} r_\sigma \end{pmatrix}. \quad (1.39)$$

We propose many ways for solving (1.33)–(1.38). The choice depends on the relative dimensions n, p and m of the variables y, z and s.

1.6.1 Case 1: p is small ($p \leq n$)

Case 1.1: $m \geq n$

We solve (1.39) by factoring the matrix directly, without any preliminary block pivot.

Case 1.2: $m \leq n$

To solve (1.39), we suggest pivoting on M. We obtain the following equivalent system

$$\begin{pmatrix} T & R \\ R^T & W \end{pmatrix} \begin{pmatrix} dz \\ d\zeta \end{pmatrix} = \begin{pmatrix} \varphi_z \\ \varphi_\zeta \end{pmatrix}. \quad (1.40)$$

with

$$\begin{aligned} T &= E\Lambda E^T + w_0 s_0^{-2}\pi\pi^T - E\Lambda A^T M^{-1} A\Lambda E^T, \\ R &= w_0 s_0^{-2}\pi - E\Lambda A^T M^{-1}\omega\sigma^{-2} f_2'(y), \\ W &= w_0 s_0^{-2} + \omega\sigma^{-2} - \omega\sigma^{-2} f_2'(y)^T M^{-1}\omega\sigma^{-2} f_2'(y) \\ \varphi_z &= r_z - E\Lambda r_s + w_0 s_0^{-2} r_{s_0}\pi + E\Lambda A^T M^{-1}(r_y + A\Lambda r_s + \omega\sigma^{-2} r_\sigma f_2'(y)), \\ \varphi_\zeta &= r_\zeta + w_0 s_0^{-2} r_{s_0} - \omega\sigma^{-2} r_\sigma + \omega\sigma^{-2} f_2'(y)^T M^{-1}(r_y + A\Lambda r_s + \omega\sigma^{-2} r_\sigma f_2'(y)). \end{aligned} \quad (1.41)$$

To compute the inverse M^{-1}, we use the Shermann-Morrison formula. Indeed, the matrix

$$B = A\Lambda^{\frac{1}{2}} + \omega^{\frac{1}{2}}\sigma^{-1}f_2'(y)$$

has fewer columns than rows and we may write

$$M^{-1} = \Delta^{-1} - \Delta^{-1}B(I + B^T\Delta^{-1}B)^{-1}B^T\Delta^{-1}.$$

The inner matrix $(I + B^T\Delta^{-1}B)$ has dimension $m \times m$ which is quite smaller than the dimension of M.

1.6.2 Case 2: p is large ($p \geq n$)

When the number p of subproblems is relatively large, the oracle returns one cut per subproblem. Therefore, m is also large, and there is no incentive for pivoting on M as in Case 1. Since the rows of E are orthoganal, the matrix $E\Lambda E^T$ is diagonal and the inverse of $E\Lambda E^T + w_0 s_0^{-2}\pi\pi^T$ can be computed explicitely by the Shermann-Morrison formula. Thus, we suggest to pivot on $E\Lambda E^T + w_0 s_0^{-2}\pi\pi^T$.

1.7 Special case: the smooth component is linear

To work with simpler formulas in Problem 1.11, we make the following substitutions:

$$(b, A) \to A, \quad (-\pi, E) \to E, \quad \begin{pmatrix}\bar{\theta}\\ c\end{pmatrix} \to c \quad \text{and} \quad \begin{pmatrix}s_0\\ s\end{pmatrix} \to s. \tag{1.42}$$

The system (1.25)-(1.26) can be simplified. The variable ζ and the constraint $\zeta \geq b^T y$ can be eliminated. The main component become

$$K'(u) = \begin{pmatrix}\rho Q(y - \bar{y}) + H'(y)\\ \pi\end{pmatrix},$$

$$K''(u) = \begin{pmatrix}\rho Q + H''(y) & 0\\ 0 & 0\end{pmatrix},$$

and

$$\frac{\partial h}{\partial u} = \begin{pmatrix}A\\ -E\end{pmatrix}.$$

Finally, we have

$$F'(\bar{s}) = -\bar{w}\bar{s}^{-1},$$

with $\bar{w} = (w_0, w)$

$$F''(s) = \bar{W}\bar{S}^{-2}.$$

The first order optimality conditions boil down to

$$\begin{aligned}\rho Q(y - \bar{y}) + b + H'(y) + AWs^{-1} &= 0,\\ EWs^{-1} &= \pi,\\ A^T y - E^T z + s &= c.\end{aligned}$$

The system to be solved takes the form

$$\begin{aligned}(\rho Q + H''(y))dy - AWS^{-2}ds &= r_y, &(1.43)\\ EWS^{-2}ds &= r_z, &(1.44)\\ A^T dy - E^T dz + ds &= r_s, &(1.45)\end{aligned}$$

with residuals defined by

$$\begin{aligned}r_y &= -(\rho Q(y-\bar{y}) + b + H'(y) + AWs^{-1}),\\ r_z &= -(-EWs^{-1} + \pi),\\ r_s &= -(A^T y - E^T z + s - c).\end{aligned}$$

The matrix associated with the linear system (1.43) is

$$N = \begin{pmatrix} \Delta & 0 & -A\Lambda \\ 0 & 0 & E\Lambda \\ A^T & -E^T & I \end{pmatrix},$$

with

$$\Delta = \rho Q + H''(y)$$

and

$$\Lambda = WS^{-2}.$$

When the oracle computes $f_1(y)$ for the first time, it returns p function values $f_{1\bullet}(y)$ and p subgradients $f'_{1\bullet}(y)$. Consequently, in practical implementation, we always have $m \geq p$. Depending on the relative values of m, n and p, we use one of the methods below to solve the system (1.43).

1.7.1 Case 1: p is small ($p \leq n$)

Case 1.1: $m \geq n$

Let us pivot on the SE diagonal block of N. We obtain the equivalent system:

$$\begin{pmatrix} \Delta + A\Lambda A^T & -A\Lambda E^T \\ -E\Lambda A^T & E\Lambda E^T \end{pmatrix} \begin{pmatrix} dy \\ dz \end{pmatrix} = \begin{pmatrix} r_y + A\Lambda r_s \\ r_z - E\Lambda r_s \end{pmatrix}. \quad (1.46)$$

We solve this system by factoring it directly, without performing a preliminary block pivot.

Case 1.2: $m \leq n$

The proximal term Q guarantees that the matrix Δ has an inverse. We assume that the inverse can be computed trivially. Successive substitutions yield

$$\begin{aligned}dz &= [E(\Lambda^{-1} + A^T\Delta^{-1}A)^{-1}E^T]^{-1}\left(r_z + E(\Lambda^{-1} + A^T\Delta^{-1}A)^{-1}(A^T\Delta^{-1}r_y - r_s)\right),\\ ds &= \Lambda^{-1}(\Lambda^{-1} + A^T\Delta^{-1}A)^{-1}(E^T dz - A^T\Delta^{-1}r_y + r_s),\\ dy &= \Delta^{-1}(A\Lambda ds + r_y).\end{aligned}$$

The crux in the computation is the solving of linear systems of the following types

$$(\Lambda^{-1} + A^T\Delta^{-1}A)v = \gamma \tag{1.47}$$

and

$$[E(\Lambda^{-1} + A^T\Delta^{-1}A)^{-1}E^T]w = \delta. \tag{1.48}$$

The square matrix $\Lambda^{-1} + A^T\Delta^{-1}A$ has dimension $m \times m$. The approach is interesting since m is small with respect to n and p. Since (1.47) is to be solved with different right-hand sides it is worth using a Cholesky factorization and perform backsolves with the Cholesky factors on the various right-hand sides.

When Δ is independent of y —a case that occurs when $H(y) = 0$— then the product $A^T\Delta^{-1}A$ keeps constant from one Newton iteration to the next. On large scale problems, this property may significantly reduce the computing time. It is also worth noticing that the transformation resulting from the addition of new cuts is incremental. When a new cut is added, A becomes (A, a) and

$$(A, a)^T\Delta^{-1}(A, a) = \begin{pmatrix} A^T\Delta^{-1}A & A^T\Delta^{-1}a \\ a^T\Delta^{-1}A & a^T\Delta^{-1}a \end{pmatrix}.$$

If $A^T\Delta^{-1}A$ is saved from one outer iteration to the next, the updating takes $m + 1$ scalar products of n-dimensional vectors while forming the product anew would require $(m + 1)^2$ scalar products.

1.7.2 Case 2: p is large ($p \geq n$)

Just as in the preceding section, let us pivot on the SE diagonal block of N. We obtain the equivalent system:

$$\begin{pmatrix} \Delta + A\Lambda A^T & -A\Lambda E^T \\ -E\Lambda A^T & E\Lambda E^T \end{pmatrix} \begin{pmatrix} dy \\ dz \end{pmatrix} = \begin{pmatrix} r_y + A\Lambda r_s \\ r_z - E\Lambda r_s \end{pmatrix}. \tag{1.49}$$

If p is relatively large compare to n, it is best to block pivot on $E\Lambda E^T$, wich is a diagonal matrix plus a symmetric rank one matrix. We obtain by substitution

$$\begin{aligned} dy &= [(\Delta + A\Lambda A^T) - A\Lambda E^T(E\Lambda E^T)^{-1}E\Lambda A^T]^{-1}(ry - A\Lambda rs + A\Lambda E^T(E\Lambda E^T)^{-1}(rz - E\Lambda rs)) \\ dz &= (E\Lambda E^T)^{-1}E\Lambda A^T dy + (E\Lambda E^T)^{-1}(rz - E\Lambda rs). \end{aligned}$$

1.8 Dual analytic center

The dual method works on points (y, z, ζ) in the primal space (sorry for the confusing use of primal and dual!). However, at the solution, or close to it, the computation of the Newton direction yields as a by-product an approximate dual analytic center as defined earlier.

Let $(y^c, z^c, \zeta^c, s_0^c, s^c)$ be the output of the computation of the generalized analytic center. This vector is an approximate solution of the first order optimality conditions 1.27 and 1.29

$$\rho Q(y^c - \bar{y}) + H'(y^c) + f_2'(y^c)(1 + w_0(s_0^c)^{-1}) + Aw(s^c)^{-1} = -r_y \approx 0.$$

Let

$$\tilde{x} = \frac{w(s^c)^{-1}}{1 + w_0(s_0^c)^{-1}} > 0.$$

If we use the perturbation vector

$$r = f_2'(y^c) + A\tilde{x}$$

in (1.27) we obtain a lower bound estimate. The residual is thus

$$r = \frac{1}{1 + w_0(s_0^c)^{-1}}(-r_y - \rho Q(y^c - \bar{y}) - H'(y^c)).$$

Note that the residual is likely to be small when y^c is good approximation to the analytic center ($r_y \approx 0$) and in the meantime is close to $\bar{y}$.

Chapter 2

Installation

OBOE is implemented in C++ and uses BLAS[8], LAPACK[6] and Lapack++ [7] for the underlying matrix operations.

Currently OBOE is supported on Linux and Windows platforms.

2.1 OBOE on Linux

The OBOE distribution comes as a compressed file in one of the 2 forms

```
OBOE-<version>.tar.gz, or
OBOE-<version>.tar.bz2
```

The user needs to untar one of the above files, for example by doing the following:

```
bunzip2 OBOE-<version>.tar.bz2, or
gunzip OBOE-<version>.tar.gz
```

and then run *tar*

```
tar -xvf OBOE-<version>.tar
```

OBOE is built and distributed using the automake [2] utility. It provides an easy-to-use framework for automatically creating makefiles, documentation, distributables. It generates a *configure* script which the user needs to run the first time. This script detects the system settings and checks for the existence of the required utitlities and libraries. It uses template makefiles Makefile.am to generate the correct makefiles for the system. The user can check the functionalities provided by *configure* by using the following command:

```
configure --help
```

2.2 OBOE on Windows

OBOE has a distributable for Windows on Visual C# .NET 2003 framework. This comes as a set of Visual C++ Projects and Visual C++ Solution files(.sln).

OBOE has been tested with Lapack++ versions 2.1.0 and 2.4.1 [1]. Since these did not have support on Windows, we have incorporated these files in the OBOE distribution. Users using the Windows version do not need to worry about getting the set of BLAS, LAPACK and LAPACKPP libraries as they are bundled with the distribution. However, it is advisable to check that the default libraries perform efficiently or the users should try to obtain the best possible libraries for their platform.

Note: Recent releases of Lapack++ version 2.4.7 and after have support for Visual C++ Projects, but we have not yet tested these releases so we leave it to the discretion of the user if they want to try the new release.

The Linux distributables can also be used on Windows if the user has MinGW[10] installed.

[1] The intermediate releases of Lapack++ had a bug which was resolved in 2.4.1.

Chapter 3

Using OBOE

Now we take a look at the software implementation of the ACCPM method in OBOE. OBOE supports optimization of the following function

$$\min\{f(y) = f_1(y) + f_2(y) \mid y \in Y_1 \cap Y_2\}, \tag{3.1}$$

where f_1 and f_2 are convex functions on R^n and $Y_i \subset R^n$, $i = 1, 2$, The function f_1 and the set Y_1 are revealed by a first order oracle, and f_2 is a convex twice continuous differentiable function. In the current implementation Y_2 can be composed of the following set of constraints:

1. Linear inequality constraints
2. Simple ball constraints $||y - y^c|| \leq R$
3. Box constraints $\beta_l \leq y \leq \beta_u$
4. Linear equality constraints of the form $\{y \mid D^T y = d\}$.

We will later see how to specify the above forms of Y_2.

3.1 C++ callable library

OBOE is distributed as a bunch of C++ libraries which need to be linked in by the user to create the final executable.

3.1.1 OBOE classes

In this section we look at the API provided with OBOE to enable users to incorporate OBOE functionality in their code. The main classes of interest to the user are

1. Oracle
2. Parameters
3. QpGenerator

Oracle. This class is designed to allow the user to specify the "oracle" for their problem. Both the non-smooth $f_1(.)$ and smooth $f_2(.)$ functions are implemented using the class **OracleFunction**. The *Oracle* object keeps a handle to the **OracleFunction** objects created by the user.

OracleFuntion. This is an abstract class designed to allow the user to specify the "oracle functions". This class can be used for providing both smooth and non-smooth function information. Let us now recall what information needs to be provided for each of the functions.

Non-smooth function. For the non-smooth function $f_1(.)$ OBOE calls the user oracle with a query point $\bar{y}$ and expects the following information

- Is the query point $\bar{y}$ feasible
- The cut information of the form

$$a^T(y - \bar{y}) + c \leq 0.$$

  Whether the cut information is of the feasibility kind($\bar{y}$ is not a feasible point) or optimality we can find a hyperplane which separates the current query point from the current localization set. This successively reduces the size of the localization set. The oracle needs to specify the values a and c. If the function $f_1(.)$ is disaggregated a is a matrix containing the cuts for each function and c a vector of compatible dimension. Otherwise a is simply a vector and c a scalar value.

The user provides this information by subclassing from *OracleFunction* and defining the *OracleFunction::eval* function. For example, the user defines the following class,

```
class MyOracleFunction : public OracleFunction {

public:
virtual int eval(const AccpmVector &y,
                   AccpmVector &functionValue,
                   AccpmGenMatrix &subGradients,
                   AccpmGenMatrix *info);
};
```

Smooth function. The *OracleFunction* class is also used to specify the smooth function. Recall that for the smooth function $f_2(.)$ the user needs to give the function value, the gradient value and the hessian at the query point $\bar{y}$. The user provides the above information via the *OracleFunction::eval* function.

Refer to `Oracle.h` to see the exact syntax and usage of *eval* function.

Parameters. The *Parameters* object is used for specifying the control parameters for OBOE. This class provides a set of functions to regulate the various parameters. Refer to `Parameters.h` to get a detailed description of the provided functions. For ease of use it also allows the user to read in a text file which can specify `int, double`,

and `string` valued parameters. The format of this file is very simple, each line contains the following
Name Type Value
where *Name* is the name of the parameter, *Type* is the type of the parameter and *Value* is the value to be assigned to the parameter. For example to set maximum number of outer iterations the user can place the following line in the parameter file
MaxOuterIterations Int 200
This will limit the number of calls to the *oracle* to 200.
The same can be done using the *Parameters* class function

```
setIntParameter("MaxOuterIterations", 200);
```

In section 3.2 we will look at the different parameters and control functions provided by the `Parameters` class.

QpGenerator. This is the main interface to OBOE. This class is responsible for generating query points and subsequently calling the *oracle*. A *QpGenerator* object is initialized by providing a handle to the above two objects, the *Oracle* object and the *Parameters* object. After initialization the user can use the *QpGenerator* object to generate query points via the *run* method. Each call of *run* method generates a query point and calls the *OracleFucntion::eval* function defined by the user's *OracleFunction* object and returns the status of these operations. Typically the user would call *run* as long as it does not return a 0 value. A 0 signals termination of the query point generation process and could be caused due to one of the following:

1. Maximum number of outer iterations reached, or
2. Relative Gap is below the specified Tolerance, or
3. User asked the QpGenerator to stop by returning a 1 in the *OracleFunction::eval* function.

3.1.2 Linear inequality constraints

The linear inequality constraints can be added as feasibility cuts by the oracle at the beginning. The oracle needs to ensure that the constraint is represented in the form

$$a^T(y - \bar{y}) + c \leq 0,$$

where $\bar{y}$ is the query point at which the oracle is called. So a linear constraint of the form

$$d^T y \leq d$$

will have to be written as

$$d^T(y - \bar{y}) + d^T\bar{y} - d \leq 0.$$

The orcale will return the values vector d and scalar value $d^T\bar{y} - d$.

3.1.3 Ball and box constraints

Box constraints Box constraints are used to specify bounds on the variables, for example $l \leq y \leq u$. The lower bound on variables can be specified by the function

```
bool Parameters::setVariableLB(const StdRealVector& v);
```

Similarly, the upper bound on variables can be specified by the function

```
bool Parameters::setVariablerUB(const StdRealVector& v);
```

Ball constraints These constraints of the form

$$||y - y^r||^2 \leq R^2$$

are specified by providing the center of the ball y^r via the function

```
bool Parameters::setCenterBall(const StdRealVector& v);
```

The radius, R, is specified via the double parameter *RadiusBall*.

3.1.4 Linear equality constraints

For some applications the y variables need to satify $\{y \mid D^T y = d\}$. These constraints can be specified by the following function

```
void Parameters::addEqualityConstraints(const AccpmGenMatrix &constraints,
const AccpmVector &rhs);
```

The matrix *constraints* should contain the matrix D and the vector *rhs* specifies the values of d.

3.2 Parameter Listing

Here we provide the various control parameters for OBOE, used for tuning the behaviour of the algorithm. The table gives the names of control parameters defined in the header file `Parameters.h`), their type, default values, and descriptions.

`ProblemName`	type: String, default: OBOE General Problem Symbolic name for the problem being solved by the user.
`OptimizationType`	type: String, default: Min Min: Minimize the objective function or Max: Maximize the objective function or
`NumVariables`	type: Integer, default: 0 Number of variables in the problem. This needs to be set by the user.

`NumProblems`	type: Integer, default: 1 Number of functions in non-smooth $f_1(.)$ function. If $f_1(.)$ is disaggregated into sum of functions this parameter specifies the number of such functions.
`MaxOuterIterations`	type: Integer, default: 1000 Limit on the number of iterations performed by OBOE. This is the miximum number of query points generated.
`MaxInnerIterations`	type: Integer, default: 50 Maximum number of inner iterations performed by OBOE. This limits the number of iterations performed to solve the Newton system internally, in every outer iteration. Typically, the average number of inner iterations is about 2-3 times the number of outer iterations.
`ObjectiveLB`	type: Double, default: ACCPM_MINUS_INF (defined in `AccpmDefs.h`) Lower bound on the objective function value.
`ObjectiveUB`	type: Double, default: ACCPM_PLUS_INF (defined in `AccpmDefs.h`) Upper bound on the objective function value.
`ComputeLowerBound`	type: Integer, default: 1 If OBOE should compute a lower bound for minimization problems internally. Otherwise, OBOE will depend on the user for providing a valid lower bound via `Oracle::getLowerBound()`. If you are not sure of how to provide a lower bound of the problem, do not change this parameter setting.
`Verbosity`	type: Integer, default: 0 Controls the amount of information provided by OBOE. 1 : Prints function value information received from the *Oracle*. 2 : Also prints the Relative Gap information. 3 and higher : Useful for debugging.
`Tolerance`	type: Double, default: 10^{-6} This specifies the convergence criterion. OBOE terminates when the Relative Gap reduces to value less than the `Tolerance`.
`Rho`	type: Double, default: 1.0 The weight on the proximal term $1/2(y-\bar{y})^T Q(y-\bar{y})$.
`DynamicRho`	type: Integer, default: 0 0 : Do not update the value of `Rho`. 1 : Update the value of `Rho` at every iteration. OBOE uses an internal strategy to control the value of parameter `Rho`.
`RhoMax`	type: Double, default: 100 If `DynamicRho` is 1 then we can limit the maximum value `Rho` can take by specifying the value of this parameter.

`RhoMin` type: Double, default: 10^{-6}
If `DynamicRho` is 1 then we can limit the minimum value `Rho` can take by specifying the value of this parameter.

`WeightEpigraphCutInit` type: Double, default: 1.0
The weight s_0 on the lograthmic barrier for the epigraph cut.
This value affects only once we have found feasible point and have optimality cuts.

`WeightEpigraphCutInc` type: Double, default: 1.0
This parameter control the amount by which the parameter `WeightEpigraphCutInit` should be increased in each iteration.
The default behaviour is to give the epigraph cut as much weight as the number of cuts.

`Ball` type: Integer, default: 0
0 : No ball constraints.
1 : There are ball constraints which need to be satisfied by the query point y.

`RadiusBall` type: Double, default: 10^5
This is the radius, R, of the ball constraint $||y - y^r||^2 \leq R^2$.
The center of the ball must be specified by the function `setCenterBall(const StdRealVector &v);`

`Filter` type: Integer, default: 1
0 : Keep duplicate cuts.
1 : If the user gives duplicate cuts, do not add them to the internal matrices.

`ConvexityCheck` type: Integer, default: 0
Check if the optimality cuts given by the user respect convexity.
Incase there is a violation, OBOE prints a warning that the objective function value is under or over estimated as the case may be.
This does not do any fixing, only warns the user of the violations if they occur.

`ConvexityFix` type: Integer, default: 0
If `ConvexityFix` is set to 1, OBOE tries to internally correct the convexity violation if it is possible.

type: , default:

The `Parameters` class(3.1.1) provides API for specifying the above parameters either in a text file or via the following function calls.

For `int` parameters the user can either specify in the input file

```
<name> I <value>
```

or use the function call:

```
bool setIntParameter(const char *name, int value);
```

The above mechanism is also used for setting `bool` parameters, with the only differ-

ence that the value can be either 0 or 1.

For `String` parameters the user can either specify in the input file

```
<name> S <value>
```

or use the function call:

```
bool setStringParameter(const char *name, const string &value);
```

For `Double` parameters the user can either specify in the input file

```
<name> D <value>
```

or use the function call:

```
bool setRealParameter(const char *name, double value);
```

3.2.1 Retrieving parameter values

The scalar parameters have `getTypeParameter` functions similar to the `set<Type>Parameter` functions.

For getting the value of `int` parameter

```
int getIntParameter(const char *name);
```

For getting the value of `double` parameter

```
double getRealParameter(const char *name);
```

For getting the value of `string` parameter

```
const string getStringParameter(const char *name);
```

3.2.2 Array parameter values

Below we describe some of the other API functionality provided by the `Parameters` class.

`bool setStartingPoint(const StdVector& v)` function can be used to set the first query point. It defaults to a zero vector.

The starting point can be retrieved by `const AccpmVector *getStartingPoint() const`.

`bool setVariableLB(const StdRealVector& v)` function can be used for specifying the lower bound on the variables. It defaults to `ACCPM_MINUS_INF`.

The variable lower bound values can be retrieved by `const AccpmVector *getVariableLB() const`.

`bool setVariableUB(const StdRealVector& v)` function can be used for specifying the upper bound on the variables. It defaults to `ACCPM_PLUS_INF`.

The variable upper bound values can be retrieved by `const AccpmVector *getVariableUB() const`.

`bool setPi(const StdRealVector& v)` function allows the user to specify the weight π on the non-smooth function $f_1(.)$. The length of the vector v should be equal to the $NumSubProblems$.

The π vector can be retrieved by `const AccpmVector *getPi() const`.

`bool setB(const StdRealVector& v)` and `bool setB(const AccpmVector& v)` allow the user to specify the linear component of the objective function. OBOE allows either the `Oracle` has a general smooth function or is completely linear in which case the vector b of $b^T y$ function is specified by the above function. If the smooth function $f_2(.)$ has a linear component in addition to a non-linear part the user should specify it in the *OracleFunction* itself.

The b vector can be retrieved by `const AccpmVector *getB() const`.

`bool setCenterBall(const StdRealVector& v)` function is useful for specifying the center y^r of the ball constraint $||y - y^r||^2 \leq R^2$.

The center of the ball constraint can be retrieved by `const AccpmVector *getCenterBall() const`.

3.3 Linear Algebra

OBOE uses LAPACK++ v. 2.0.x for underlying linear algebra operations.

LAPACK++ [7] is a library for high performance linear algebra computations. It wraps BLAS and LAPACK functionality in C++ classes. It is provided under Lesser GPL licensing scheme. Note: There were some errors in post 2.1 release which were fixed in 2.4.1. OBOE has been tested with 2.1.0 and 2.4.2. We have not yet tested the recent releases so it is upto the user if they want to get a recent release of Lapack++.

The BLAS and LAPACK libraries have a huge impact on the performance of OBOE. We recommend the user to obtain the most efficient BLAS and LAPACK libraries for their system. Fine tuned blas and lapack libraries are provided by ATLAS [1], which aims at creating highly efficient libraries depending on the system parameters.

3.4 Serialization

OBOE has support for serialization on Linux platform. It uses the serialization libraries provided by boost

$$http://www.rrsd.com/boost/libs/serialization/doc/index.html$$

By default the support for these libraries is turned off, but can be enabled by

```
configure --enable-serialization=yes
```

These libraries enable the save and load facilty for OBOE, incase the user wants to save the state and restart OBOE.

Chapter 4

Benchmark for OBOE

4.1 Introduction

In this section we solve a set of problems using OBOE. The objective here is to illustrate the performance and robustness of OBOE. Also, we would like to show the wide variety of problems that fit within the nondifferentiable convex optimization framework.

- The cutting stock problem
- The quadratically constrained quadratic programming problem
- The traveling salesman problem
- Linear support vector machine

4.2 The cutting stock problem

4.2.1 Description of the problem

The cutting stock problem[4] arises from many physical applications in industry. For instance, in the *paper cutting division* of a given *paper mill*, there is a number of rolls of paper of fixed width waiting to be cut, and different manufacturers want different numbers of rolls of various-sized widths. The problem here is to find how to cut the rolls so that the least amount of left-overs are wasted. This problem can be formulated as an integer linear programming problem[3]

4.2.2 Mathematical formulation

Consider the previous *paper cutting* instance, each valid cutting of a roll is called a *pattern*. For each width, we have to produce the quantity ordered. A *pattern* gives the number of pieces for each width involved, and there are several possible patterns for a roll.

We can consider the problem to be an integer linear program, where the variables are the number of each pattern to cut. The objective function is to minimize the number

of rolls that are cut. The constraints in the problem require that we cut enough rolls with certain patterns to fulfill the orders.

Let a_{ij} be the number of times the order width i is produced in pattern j. Let x_j be the number of times the pattern j is used. Let b_i be the demand for the width i, and m is the number of such demands. Let n be the number of patterns in the model. The problem can be formulated as [3] :

$$\begin{cases} \min & \sum_{j=1}^{n} x_j \\ \text{subject to} & \sum_{j=1}^{n} a_{ij} x_j \geq b_i \\ & x_j \geq 0, \end{cases} \tag{4.1}$$

with the initial set of patterns. Since this set of patterns may not immediately gives the solution of the problem, we need to generate new patterns to get better solution.

4.2.3 Solving with OBOE

To solve the problem 4.1 we consider the LP dual of the above problem:

$$\begin{cases} \max & \sum_{i=1}^{m} y_i b_i \\ \text{subject to} & \sum_{i=1}^{m} a_{ij} y_i \leq 1 \\ & y_i \geq 0. \end{cases} \tag{4.2}$$

As mentioned before we do not have all the patterns to start with. We need to generate new patterns to reduce the number of rolls used. To generate these patterns we solve another optimization problem such that we get a pattern with negative reduce cost while ensuring that it does not exceed the width of the roll L. For this purpose, the dual variables of problem (4.1) are used in the *knapsack problem* below :

$$\begin{cases} \max & Z = \sum_{i=1}^{m} y_i z_i \\ \text{subject to} & \sum_{i=1}^{m} w_i z_i \leq L \\ & z_j \text{ integer}. \end{cases} \tag{4.3}$$

Both the main linear problem (4.1) and the associated knapsack (4.3) are solved consecutively until there are no more patterns to include in the main problem. For more details refer to website:

`http://www-fp.mcs.anl.gov/otc/Guide/CaseStudies/cutting/math.html`

If the knapsack problem 4.3 returns a value $Z \leq 1$, the oracle returns an optimality cut otherwise we return a feasibility cut which is the new pattern, z generated as the solution to the knapsack problem.

4.3 The quadratically constrained quadratic programming problem

4.3.1 Description of the problem

The quadratic problem of our concern here can be expressed as follows:

$$\begin{cases} \min & \sum_{j=1}^{n}(a_j y_j - b_j)^2 \\ \text{subject to} & \sum_{j=1}^{n}(c_{ij} y_j - e_{ij})^2 \leq f_i, i = 1, 2, \cdots, m \\ & Ay \leq \ell. \end{cases} \tag{4.4}$$

Problem (4.4) extend the standard one commonly encountered the litterature (see http://www.numerical.rl.ac.uk/qp/qp.html) in the following form :

$$\begin{cases} \min & \frac{1}{2} y^T Q y \\ \text{subject to} & Ay \leq L \\ & Ey = G. \end{cases} \tag{4.5}$$

Dealing with quadratic constraints is one of the advantage of the cutting plane method. Moreover, we can exploit the feature of disaggregation to speed the optimization process, since the objective function is structurally disaggregated (and also separable in this case).

4.3.2 Solving with OBOE

Using OBOE to solve the QP, the *Oracle* checks whether or not the query point is feasible.

Optimality Cuts

For query point $\bar{y}$ if

$$\sum_{j=1}^{n}(c_{ij} y_j - e_{ij})^2 \leq f_i, i = 1, 2, \cdots, m$$

then we give optimality cuts using the convexity of the objective function If we use the disaggregated form and consider $f_1(.)$ to be sum of n functions, we get the following cuts

$$(2a. * (a. * y - b)). * (y - \bar{y}) + (a. * y - b)^2 \leq 0,$$

where $.*$ is the component-wise vector multiplication.

Feasibility Cuts

For query point $\bar{y}$ if for constraint i

$$\sum_{j=1}^{n}(c_{ij} y_j - e_{ij})^2 \geq f_i$$

then we give feasibility cuts using the convexity of the constraints. For each constraint i which is not feasible with respect to $\bar{y}$ we have a feasibility cut given by

$$(2c_i. * (c_i. * y - e_i))^T (y - \bar{y}) + (c_i. * y - e_i)^2 - f_i \leq 0,$$

where $.*$ is the component-wise vector multiplication.

4.4 The traveling salesman problem

4.4.1 Description of the problem

An illustrative defintion of the traveling salesman problem, or TSP for short, is this: given a finite number of "cities" along with the cost of travel between each pair of them, find the cheapest way of visiting all the cities and returning to your starting point. For more detailed information on TSP problem we refer to the website

`http://www.math.princeton.edu/tsp/index.html`

4.4.2 Algebraic description

Consider a weigthed graph $G = (V, A, W)$, where V is the set of vertex, $A \subset V \times V$ the set of arcs, and $W : u \in A \rightarrow W(u) \in R$ the weight function. The *Traveling Salesman Problem* (TSP) consist in finding a *hamiltonian cycle* in G with a minimum cost. A hamiltonian cycle in G can be defined by a bijection $\sigma : V \rightarrow V$ such that

$$\forall v \in V, (v, \sigma(v)) \in A. \tag{4.6}$$

Given an hamiltonian cycle in G defined by σ, the associated cost is defined by

$$C(G, \sigma) = \sum_{v \in V} W((v, \sigma(v)). \tag{4.7}$$

The problem is known to be NP-Complete[5] and has many significant applications including those in transport and logistics.

4.4.3 Mathematical formulation

Given the **symmetric** cost matrix $C = (c_{ij})$ of the graph G of order n, the TSP can be formulated as follows:

$$\begin{cases} \min & \sum_{i,j} x_{ij} c_{ij} \\ \text{subject to} & \sum_{j=1}^{n} x_{ij} = 2, \forall i \in \{1, \cdots, n\} \\ & x_{ij} \in \{0, 1\} \end{cases} \tag{4.8}$$

A straightforward relaxed formulation is :

$$\left\{ \max_{\lambda} \min_{x \in \mathcal{B}(G)} \sum_{i,j} x_{ij} c_{ij} + \sum_{i=1}^{n} \lambda_i (\sum_{j=1}^{n} x_{ij} - 2) \right. \tag{4.9}$$

which is equivalent to

$$\left\{ \max_{\lambda} \min_{x \in \mathcal{B}(G)} \quad \sum_{i=1}^{n} [-2\lambda_i + \frac{1}{2} \sum_{j=1}^{n} x_{ij}(c_{ij} + \lambda_i + \lambda_j)] \right. \tag{4.10}$$

where $\mathcal{B}(G)$ is a one-tree built over the cost matrix defined by

$$C_{ij}^{(\lambda)} = c_{ij} + \lambda_i + \lambda_j. \tag{4.11}$$

Since $\mathcal{B}(G)$ is not equivalent to the set of tours in G, problem (4.10) gives a lower (and sometimes upper) bound of the original problem, which can then be solved using the branch and bound paradigm.

4.5 Linear support vector machine

4.5.1 Description of the problem

The purpose of linear separation is to find a linear function to separate multi-attribute instances that are partitioned into two subsets. The goal of the linear separation is to find a linear form in the space of the attributes that leaves the two subsets on opposite sides. In general it is not possible to achieve perfect separation. One has to look for a surrogate objective. A natural one is to minimize the total number of misclassified instances. Unfortunately this leads to a mixed integer programming problem, which may be very hard even for moderate size data. A more tractable approach [9] consists of minimizing the total deviation (or gap) of the misclassified instances. This problem is a convex non-differentiable problem for which there exists solution methods with polynomial complexity estimates.

4.5.2 Mathematical formulation

Given a set of points $\mathcal{A} = \{a_i \in \mathcal{R}^n, i = 1, 2, \cdots, N\}$, and a partition $\mathcal{S}_1 \cup \mathcal{S}_2$ of the set of indices $\mathcal{S} = \{1, 2, \cdots, N\}$, we wish to find $w \in \mathcal{R}^n$ and $\gamma \in \mathcal{R}$ such that the hyperplane $\{x \mid w^T x = \gamma\}$ separates the two subsets $\mathcal{A}(\mathcal{S}_1)$ and $\mathcal{A}(\mathcal{S}_2)$, where

$$\mathcal{A}(\mathcal{S}_1) = \{a_i \in \mathcal{A} \mid i \in \mathcal{S}_1\}, \tag{4.12}$$

$$\mathcal{A}(\mathcal{S}_2) = \{a_i \in \mathcal{A} \mid i \in \mathcal{S}_2\}. \tag{4.13}$$

For typographical convenience, we will write (w, γ) instead of (w^T, γ).

Actually, one looks for a strong separation. Thus, given a *separation margin* $\nu > 0$, we hope to achieve the separation properties (4.14-4.15) displayed bellow

$$\forall a_i \in \mathcal{A}(\mathcal{S}_1) \quad w^T a_i \geq \gamma + \nu, \tag{4.14}$$

$$\forall a_i \in \mathcal{A}(\mathcal{S}_2) \quad w^T a_i \leq \gamma - \nu. \tag{4.15}$$

In general, there is no guarantee that the two sets can be strongly separated. Therefore, for any choice of w and γ, we might observe *misclassification errors*, which we define as follows

$$e_i^1 = \max(-w^T a_i + \gamma + \nu, 0), \quad i \in \mathcal{S}_1, \tag{4.16}$$

$$e_i^2 = \max(w^T a_i - \gamma + \nu, 0), \quad i \in \mathcal{S}_2. \tag{4.17}$$

The linear separation problem can be formulated as the following minimization problem in $\mathcal{R}^{\setminus+\infty}$:

$$\min_{(\omega,\gamma)\in\mathcal{R}^{\setminus}\times\mathcal{R}} F(\omega,\gamma) = \frac{1}{|\mathcal{S}_1|}\sum_i e_i^1 + \frac{1}{|\mathcal{S}_2|}\sum_i e_i^2 \tag{4.18}$$

4.5.3 Linear separation oracle

Let $\mathcal{A}_1'$ and $\mathcal{A}_2'$ define the set of misclassified points for $\mathcal{A}(\mathcal{S}_1)$ and $\mathcal{A}(\mathcal{S}_2)$ respectively. Hence,

$$\begin{aligned}\mathcal{A}_1' &= \{i : a_i \in \mathcal{A}(\mathcal{S}_1), e_i^1 > 0\},\\ \mathcal{A}_2' &= \{i : a_i \in \mathcal{A}(\mathcal{S}_2), e_i^2 > 0\}.\end{aligned}$$

Differentiating (4.18) in (ω,γ) yields $(-a_i, 1)$ for the misclassified points of $\mathcal{A}(\mathcal{S}_1)$, and $(a_i, -1)$ for the misclassified points of $\mathcal{A}(\mathcal{S}_2)$. Thus, for any feasible point (ω,γ), the gradient for the oracle is defined by $g = g_1 + g_2$, where

$$\begin{aligned}g_1 &= \tfrac{1}{|\mathcal{A}_1|}\textstyle\sum_{i\in\mathcal{A}_1'}(-a_i, 1),\\ g_2 &= \tfrac{1}{|\mathcal{A}_2|}\textstyle\sum_{i\in\mathcal{A}_2'}(a_i, -1).\end{aligned}$$

For details on the use of ACCPM for solving the separation problem with more general quadratic separation functions, we refer to [11].

Bibliography

[1] *Atlas*. Available at the web http://math-atlas.sourceforge.net.

[2] *Automake*. A tool for automatically generating Makefiles files compliant with the GNU Coding Standards. For details refer to: http://sources.redhat.com/automake/.

[3] V. Chvatal, *Linear Programming*, Series of Books in the Mathematical Sciences, W. H. Freeman Compagny, 1983.

[4] Z. Degrave and M. Peeters, *Benchmark results for the cutting stock and bin packing problem*, Tech. Rep. Research Report No 9820, Quantitative Methods Group, Louvain, Belgique, 1998.

[5] M. R. Garey and D. S. Johnson, *Computer and Intractability - A Guide to the Theory of NP-Completeness*, W. H. Freeman, New-York, USA, 1979.

[6] *Lapack*. Linear algebra library based on BLAS. For details refer to http://www.netlib.org/lapack.

[7] *Lapack++ library.* Available at http://www.sourceforge.net/projects/lapackpp.

[8] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh, *Blas, basic linear algebra subprograms for fortran.* Applications for Computing Machinery Transactions on Mathematical Software, 1979. For details refer to http://www.netlib.org/blas.

[9] O. L. Mangasarian, W. N. Street, and W. H. Wolberg, *Breast cancer diagnosis and prognosis via linear programming*, Operations Research, 43 (1995), pp. 570–577.

[10] *Mingw32*. Free compiler and shell system for Windows. Available at the web http://www.mingw.org.

[11] O. Peton, N. Sawhney, and J. P. Vial, *Linear and nonlinear discrimination via the analytic center cutting plane method*, To be published Optimization Methods and Software, (2006).