A Simplex-active-set Algorithm for Piecewise Quadratic Programming †

R. T. Rockafellar* and J. Sun**

Abstract

An algorithm for monotropic piecewise quadratic programming is developed. It converges to an exact solution in finitely many iterations and can be thought of as an extension of the simplex method for convex programming and the active set method for quadratic programming. Computational results show that solving a piecewise quadratic program is not much harder than solving a quadratic program of the same number of variables. In other words, the computation time is not sensitive to the increase of "pieces".

Key words: Active Set Methods, Computational Experiment, Linear Constraints, Monotropic Programming, Optimization, Piecewise Quadratic Programming, Simplex Methods.

Abbreviated title: Algorithm for Piecewise quadratic programming

† This research was supported in part by grants from the Air Force Office of Scientific Research and the National Science Foundation at the University of Washington, Seattle.

* Department of Applied Mathemati c s, University of Washington, Seattle, WA 98195

** Department of Industrial Engineering and Management Sci e nces, Northwestern University, Evanston, IL 60208

1. Introduction

In a previous paper [9] we studied the basic theory and some applications of monotropic piecewise quadratic programming (PQP for short). This model has promising applications in such areas as stochastic programming [8], optimal control [7], decomposition algorithms of quadratic programming and approximate methods of monotropic programming, etc. [9]. In addition, it is worth mentioning that in general the dual of convex quadratic programming can be reduced to this form. Due to the broad range of its applications, it is natural to ask for the development of efficient algorithms. In this paper we would like to introduce a hybric algorithm and present some initial computational results. The algorithm combines the features of simplex methods for convex programming [2][6][10] and active set methods for quadratic programming ([1] and references therein) and directly deals with the PQP model without reformulation.

The mathematical statement of the PQP problem is as follows:

$$\text{(PQP)} \begin{cases} \text{minimize} & F(x) = \sum_{j=1}^{n} f_j(x_j) \\ \text{subject to} & Ax = b \\ & c^- \le x \le c^+ \end{cases}$$

where $A \in R^{m\times n}, b \in R^m, c^-, c^+$ *and* $x \in R^n$. The vector inequality is understood coordinatewise. Each f_j is convex piecewise quadratic, i.e.

$$f_j(x_j) = \begin{cases} +\infty, & \text{if } x_j < c_j^- := c_{j0} \\ \frac{1}{2}p_{j1}x_j^2 + q_{j1}x_j + r_{j1}, & \text{if } c_{j0} \le x_j \le c_{j1} \\ \cdots & \\ \frac{1}{2}p_{jk_j}x_j^2 + q_{jk_j}x_j + r_{jk_j}, & \text{if } c_{jk_j-1} \le x_j \le c_{jk_j} \\ +\infty, & \text{if } x_j > c_j^+ := c_{jk_j} \end{cases}$$

where for each j, we call the numbers $c_{j0}, c_{j1}, \ldots, c_{jk_j}$ the *breakpoints* of f; they satisfy

$$-\infty \le c_{j0} \le c_{j1} \le \ldots \le c_{jk_j} \le +\infty.$$

Notice that the (effective) domain of the objective in problem (PQP) is a box (hyperrectangle) in R^n and is the union of finitely many smaller boxes on each of which the objective is quadratic (or linear). An intuitive idea would be to find out suboptima in all such small boxes, which corresponds to solving a series of quadratic programs, and then to choose the optimal solution among them. This is of course unrealistic because of the possible huge number of these small boxes. Nevertheless, it suggests that if we can effectively combine the two tasks — looking for the small box where the optimum resides and finding the optimum in the small box — then the algorithm will be finitely convergent.

The algorithm to be introduced below is performed on the simplex tableaux and can take advantage if most f_j's in the objective are piecewise linear. The algorithm converges to an exact solution in finitely many steps. In the following, the framework of a class of algorithms is presented in Section 2; its implementation is discussed in Section 3; the degeneracy processing is the topic of Section 4 and some computational results are reported in Section 5.

2. Framework of a Finitely Convergent Algorithm

In this section we introduce a model algorithm and clarify its convergence.

Definition (2.1) Let $\bar{x}$ be a feasible solution of PQP. Denote by $IM(\bar{x})$ the index set

$$\{i \in \{1, 2, \cdots, n\} | \bar{x}_i \text{ is a breakpoint of } f_i\}.$$

Then $\bar{x}$ is called a *quasioptimal solution* to PQP if it is the optimal solution to the following subproblem:

$$(\text{SP1})\begin{cases}\text{minimize} & F(x)\\ \text{subject to} & Ax=b\\ & c^-\le x\le c^+\\ & x_j=\bar{x}_j \quad \forall\, j\in IM(\bar{x}).\end{cases}$$

If $\bar{x}$ is a quasioptimal solution, then $F(\bar{x})$ is called a *quasioptimal value.*

According to this definition, the optimal solutions to PQP must be quasioptimal. In addition there is only a finite number of quasioptimal values. To see this point, notice that the total number of different sets $IM(\bar{x})$ is finite as $\bar{x}$ ranges over all feasible solutions to the given problems. Hence there is only a finite number of different subproblems (SP1). Hence if we can strictly descend from one quasioptimal solution to another, then after finitely many steps we will end up with an optimal solution.

The framework of this type of algorithms consists of three steps:

Algorithm (2.2).
Step 0. Find a feasible solution or determine the infeasibility of the PQP problem.
Step 1. Move from a feasible solution to a quasioptimal solution without increasing the objective and go to Step 2.
Step 2. Verify whether this quasioptimal solution is optimal. If not, find a descent direction , decrease the objective function and go to Step 1.

If each step takes finitely many operations, then any algorithm which can be embedded in this framework will have to converge in finitely many iterations.

3. Implementation

In this section we consider the question of how to realize the model algorithm by using simplex tableaux.

3.1 Initialization

Step 0 can be accomplished by any linear programming algorithm if we replace the objective function by zero. Since some efficient algorithms for piecewise linear programming have been recently proposed (e.g. [2][3][4]), a better idea is to use a piecewise linearized approximation of the objective function in order to get a starting point closer to the optimum of (PQP). Actually in certain case a quantitative estimate can be made about the distance between the optimal solutions of the piecewise linear approximation and the optimal solutions of (PQP), as indicated by the following proposition.

Proposition (3.1.1). Suppose that $c^- > -\infty$ and $c^+ < \infty$ for all j. Let $\bar{f}_j(x_j)(j=1,...,n)$ be defined as

$$\bar{f}_j(x_j)=\begin{cases}+\infty, & \text{if } x_j < c_j^- = g_{j0};\\ f_j(g_{j0})+f_j'(\frac{g_{j0}+g_{j1}}{2})(x_j-g_{j0}), & \text{if } g_{j0}\le x_j\le g_{j1};\\ \cdots & \\ f_j(g_{j,m_j-1})+f_j'(\frac{g_{j,m_j-1}+g_{jm_j}}{2})(x_j-g_{j,m_j-1}), & \text{if } g_{j,m_j-1}\le x_j\le g_{jm_j}\\ +\infty, & \text{if } x_j > c_j^+ = g_{jm_j}.\end{cases}$$

This is a piecewise linear approximation of $f_j(x_j)$. Suppose that the function $f_j(x_j)$ can be expressed as $\frac{1}{2}p_{ji}x_j^2+q_{ji}x_j+r_{ji}(p_{ji}>0)$ between the grid points $g_{j,i-1}$ and g_{ji} for $i=1,...,m_j$ and $j=1,...,n$. Then the distance between the optimal solution x^* of $\bar{F}(x)=\sum_{j=1}^n \bar{f}_j(x_j)$ and the optimal solution x^{**} of $F(x)=\sum_{j=1}^n f_j(x_j)$ on the set $S=\{x\in R^n|Ax=b, c^-\le x\le c^+\}$ satisfies $||x^*-x^{**}||\le Md$, where $||\cdot||$ is the Euclidean norm of R^n, $d=\max\{g_{ji}-g_{j,i-1}|i=1,...,m_j, j=1,...,n\}$, and M is a constant independent from d.

Proof. Since S is a compact set, both x^* and x^{**} exist. Let

$$\lambda = \min\{p_{ji} | i = 1, ..., m_j, j = 1, ..., n\}.$$

By the assumption that f_j is strictly quadratic we have $\lambda > 0$. It is not hard to see that geometrically the graph of $\bar{f}_j$ is the brokenline connecting $(g_{j0}, f_j(g_{j0})), (g_{j1}, f_j(g_{j1})), \cdots$, and $(g_{jm}, f_j(g_{jm}))$. Thus we have

$$\begin{aligned}
&\bar{F}(x^*) - F(x^{**}) \geq F(x^*) - F(x^{**}) \\
=&\sum_{j=1}^{n}(f_j(x_j^*) - f_j(x_j^{**})) \geq \sum_{j=1}^{n} f_j'(x_j^{**}, x_j^* - x_j^{**}) + \frac{1}{2}\lambda\sum_{j=1}^{n}(x_j^* - x_j^{**})^2 \\
=&F'(x^{**}, x^* - x^{**}) + \frac{1}{2}\lambda||x^* - x^{**}||^2
\end{aligned}$$

where $f'(\cdot,\cdot)$ and $F'(\cdot,\cdot)$ are the directional derivatives. Since $x^* - x^{**}$ is an ascent direction of F at x^{**},

$$F'(x^{**}, x^* - x^{**}) \geq 0.$$

Therefore

$$\bar{F}(x^*) - F(x^{**}) \geq \frac{\lambda}{2}||x^* - x^{**}||^2.$$

On the other hand, we have

$$\bar{F}(x^*) - F(x^{**}) \leq \bar{F}(x^{**}) - F(x^{**}) = \sum_{j=1}^{n}[\bar{f}_j(x_j^{**}) - f_j(x_j^{**})].$$

Assume that $g_{j,i_j-1} \leq x_j^{**} \leq g_{ji_j}$, for $j = 1, ..., n$. Then

$$\begin{aligned}
&\sum_{j=1}^{n}[\bar{f}_j(x_j^{**}) - f_j(x_j^{**})] \\
=&\sum_{j=1}^{n}\{f_j(g_{ji_j-1}) + [p_{ji_j}(\frac{g_{ji_j-1} + g_{ji_j}}{2}) + q_{ji_j}](x_j^{**} - g_{ji_j-1}) \\
&- [f_j(g_{ji_j-1}) + (p_{ji_j}g_{ji_j-1} + q_{ji_j})(x_j^{**} - g_{ji_j-1}) + (\frac{1}{2}p_{ji_j}(x_j^{**} - g_{ji_j-1})^2]\} \\
=&\sum_{j=1}^{n} p_{ji_j}(x_j^{**} - g_{ji_j-1})(\frac{g_{ji_j-1} + g_{ji_j}}{2} - x_j^{**}) \leq \sum_{j=1}^{n} p_{ji_j}(q_{ji_j-1} - g_{ji})^2/16 \\
\leq&\frac{\mu}{16}d^2 (\mu > 0)
\end{aligned}$$

where $\mu = \max\{\sum_{j=1}^{n} p_{ji} | i_j = 1, ..., m_j, j = 1, ..., n\}$.
Thus

$$\frac{\lambda}{2}||x^* - x^{**}||^2 \leq \frac{\mu}{16}d^2$$

i.e.,

$$||x^* - x^{**}|| \leq (\frac{\mu}{8\lambda})^{1/2}d.$$

$(\mathcal{Q}.\mathcal{E}.\mathcal{D}.)$

Proposition (3.1.1) says by choosing relatively fine grid points one can solve (PQP) approximately. The trade-off is that we will have to solve a piecewise linear program with many pieces. However, if we just want a good initial point for (PQP), then using the breakpoints of f_j as the breakpoints of $\bar{f}_j$ is often a good compromise.

3.2 Algorithm for Finding a Quasioptimal Solution

Now we discuss Step 1. Suppose that $\bar{x}$ is a feasible solution. In order to get a quasioptimal solution, we need to solve

$$(\text{SP1}) \begin{cases} \text{minimize} & F(x) \\ \text{subject to} & Ax = b \\ & c^- \le x \le c^+ \\ & x_j = \bar{x}_j, \ \forall \ \ j \in IM(\bar{x}). \end{cases}$$

This is an PQP problem. Let $\bar{c}^-(\bar{x})$ and $\bar{c}^+(\bar{x})$ be the closest lower and upper breakpoint vectors for $\bar{x}$, that is

$$\bar{c}^-(\bar{x}) = [\bar{c}_1^-(\bar{x}), \cdots, \bar{c}_n^-(\bar{x})]^T, \bar{c}^+(\bar{x}) = [\bar{c}_1^+(\bar{x}), \cdots, \bar{c}_n^+(\bar{x})]^T,$$

where for $j = 1, \cdots, n$,

$$\bar{c}_j^-(\bar{x}) = \max\{c_{jl} | c_{jl} \le \bar{x}_j \ \ l = 0, 1, \cdots, k_j\}$$
$$\bar{c}_j^+(\bar{x}) = \min\{c_{jl} | c_{jl} \ge \bar{x}_j \ \ l = 0, 1, \cdots, k_j\}$$

(Recall that the c_{jl}'s are the breakpoints of f_j; c_{j0} is c_j^- and c_{jk_j} is c_j^+.) Both $\bar{c}^-(\bar{x})$ and $\bar{c}^+(\bar{x})$ depend on $\bar{x}$. Then we have the following.

Proposition (3.2.1). Any optimal solution x^* to

$$(\text{SP2}) \begin{cases} \text{minimize} & F(x) \\ \text{subject to} & Ax = b \\ & \bar{c}^-(\bar{x}) \le x \le \bar{c}^+(\bar{x}) \end{cases}$$

is a quasioptimal solution to (PQP) and satisfies $F(x^*) \le F(\bar{x})$, where $\bar{x}$ is any fixed feasible solution.

Proof. Since x^* is optimal to (SP2) but $\bar{x}$ is merely feasible, we trivially have $F(x^*) \le F(\bar{x})$. Observe that x^* is a local minimizer of the problem

$$(\text{SP3}) \begin{cases} \text{minimize} & F(x) \\ \text{subject to} & Ax = b \\ & c^- \le x \le c^+ \\ & x_j = x_j^*, \ \ \forall \ j \in IM(x^*), \end{cases}$$

so by convexity it is also a global minimizer of (SP3), and hence it is quasioptimal by definition.

($\mathcal{Q.E.D.}$)

Proposition (3.2.1) says we can get a quasioptimal solution by solving a quadratic program. Yet we can make the procedure even simpler if we have a simplex tableau of $Ax = b$ on hand. To simplify the statement of algorithms, let us first introduce some terminology.

Suppose that $\tilde{c}^- = [\tilde{c}_1^-, \cdots, \tilde{c}_n^-]^T$ and $\tilde{c}^+ = [\tilde{c}_1^+, \cdots, \tilde{c}_n^+]^T$ are two vectors whose jth components are certain breakpoints of f_j. They satisfy $\tilde{c}^- \le \tilde{c}^+$. Call the set

$$[\tilde{c}^-, \tilde{c}^+] = \{x | \tilde{c}^- \le x \le \tilde{c}^+\}$$

a *piece* if for $j = 1, \cdots, n$ there is no other breakpoint of f_j between $\tilde{c}_j^-$ and $\tilde{c}_j^+$. Given a feasible solution $\bar{x}$ and a feasible direction vector $\bar{y}$ at $\bar{x}$(i.e. $\bar{x} + \epsilon\bar{y}$ is feasible for some $\epsilon > 0$), we say that the piece $[\tilde{c}^-, \tilde{c}^+]$ is associated with $(\bar{x}, \bar{y})$ if there exists a positive number $\bar{\epsilon}$ such that for all ϵ in $[0, \bar{\epsilon}]$, $\bar{x} + \epsilon\bar{y}$ is in $[\tilde{c}^-, \tilde{c}^+]$.

Notice that $\tilde{c}^-$ and $\tilde{c}^+$ depend on $\bar{x}$ and $\bar{y}$, and once $\bar{x}$ and $\bar{y}$ are given, it is not hard to determine the unique piece that is associated with them, if we adopt the regulation that $\bar{y}_j = 0$ and $\bar{x}_j = c_{jk} \Rightarrow \tilde{c}_j^- = \tilde{c}_j^+ = c_{jk}$.

Definition (3.2.2). Given $\bar{x}$ and $\bar{y}$, the procedure for finding a minimizer of $F(\bar{x}+\epsilon\bar{y})$ in the piece associated with $(\bar{x}, \bar{y})$ is called *one-piece line search from $\bar{x}$ in the direction of $\bar{y}$*, or is simply called "the one-piece line

search with $(\bar{x}, \bar{y})$". The procedure of finding the global minimizer of $F(\bar{x} + \epsilon\bar{y})$ is called *multi-piece line search from $\bar{x}$ in the direction of $\bar{y}$*, or "the multi-piece line search with $(\bar{x}, \bar{y})$".

For any $\bar{x}$ and $\bar{y}$, $F(\bar{x}+\epsilon\bar{y})$ is a piecewise quadratic function of ϵ. Suppose $F(x)$ is $\sum_{j=1}^{n} \frac{1}{2}(p_j x_j^2+q_j x_j+r_j)$ on $[\tilde{c}^-, \tilde{c}^+]$ (if $\tilde{c}_j^- = \tilde{c}_j^+$, simply take $p_j = q_j = 0, r_j = f_j(\tilde{c}_j^-)$). We make the following convention: our one-piece line search always ends up at a relative boundary point of $[\tilde{c}^-, \tilde{c}^+]$ if this point is one of the minimizers. Namely, if $\bar{y}$ is an ascent vector (i.e. $F'(\bar{x}, \bar{y}) > 0$), then the one-piece line search stops at $\bar{x}$; otherwise the one-piece line search will give $\bar{x} + \epsilon_0\bar{y}$, where

$$\epsilon_0 = \min \begin{cases} \frac{\tilde{c}_j^+ - \bar{x}_j}{\bar{y}_j} & \text{for } j \text{ with } \bar{y}_j > 0; \\ \frac{\tilde{c}_j^- - \bar{x}_j}{\bar{y}_j} & \text{for } j \text{ with } \bar{y}_j < 0; \\ \frac{-\sum_{j=1}^{n} \bar{y}_j(p_j\bar{x}_j+q_j)}{\sum_{j=1}^{n} \bar{y}_j p_j^2} & (\text{or } +\infty, \text{ if all } \bar{y}_j p_j^2 = 0). \end{cases}$$

If $\epsilon_0 = +\infty$ and $F'(\bar{x}, \bar{y}) < 0$, this means that piec e $[\tilde{c}^-, \tilde{c}^+]$ contains the half line $\bar{x} + \epsilon\bar{y}$ $(\epsilon \geq 0)$ and along this half line, $F(x)$ is linear and decreasing. In this case, the one-piece line search halts.

The algorithm for multi-piece line search consists of repeated use of the one-piece line search. Specifically we can express this as follows, where $\inf(SP4)$ stands for the infimum of problem (SP4).

Algorithm (3.2.3).
Step 1. Do one-piece line search starting with a feasible $\bar{x}$ and a vector $\bar{y}$. If ϵ_0 is $+\infty$, then $\inf(SP4) = -\infty$; stop. Otherwise go to Step 2.
Step 2. If $\epsilon_0 > 0$ but $\bar{x}+\epsilon_0\bar{y}$ is not on the relative boundary of $[\tilde{c}^-, \tilde{c}^+]$, replace $\bar{x}$ by $\bar{x}+\epsilon_0\bar{y}$; stop. Otherwise replace $\bar{x}$ by $\bar{x} + \epsilon_0\bar{y}$; go to Step 3.
Step 3. Check if $\bar{y}$ is a descent vector at $\bar{x}$. If yes, go to Step 1; if no, stop.

The Algorithm for Finding Quasistationary Solutions

Now we are ready to state our algorithm for finding quasistationary solutions. The basic idea is, instead of solving the quadratic program (SP2) that corresponds to a given $\bar{x}$, we first make a series of pivoting steps and one-piece line searches to reduce the dimension of (SP2) until (SP2) becomes a strictly convex quadratic program with minimal dimension. Then we treat (SP2) as if it is an unconstrained quadratic program with respect to the variables x_j, $j \notin IM(\bar{x})$. We take the Newton direction vector as $\bar{y}$ and do one-piece line search with $(\bar{x}, \bar{y})$. The whole procedure (reduction of dimension and one-piece line search) will be repeated until the one-piece line search no longer ends up with a point on the relative boundary of $[\tilde{c}^-, \tilde{c}^+]$. This is much like the first step of the classical "active set method" of quadratic programming, with the concept of "active constraints" being replaced by that of "breakpoints". To be sure that the algorithm can choose one quasistationary solution when there are infinitely many of them, we need the following convention.

Breakpoint Convention: If $f_j(x_j)$ is linear on $(-\infty, +\infty nfty)$, then zero is regarded as a breakpoint of f_j .

Algorithm (3.2.4).

Step 0. Suppose that $\bar{x}$ is a feasible solution and that $x_B = Tx_N + d$ is a simplex tableau of the system $Ax = b$. Partition the set B into $FB \cup IB$, where FB(the set of *free* basic indices)$= B\backslash IM(\bar{x})$ and IB(the set of *imprisoned* basic indices) $= B \cap IM(\bar{x})$. Likewise partition $N = FN \cup IN$(the sets of free and imprisoned nonbasic indices). Let T_{FI} be the submatrix of T consisting of t_{ij} with $i \in FB$ and $j \in IN$. Similarly define T_{IF}, T_{II} and T_{FF}. (See Figure 1.)
Step 1. If there is a nonzero $t_{ij} \in T_{IF}$, pivot on t_{ij} and repeat this until either $T_{IF} = \emptyset$ or $T_{IF} = 0$ (see Figure 1). Go to Step 2.

Step 2. If $FN = \emptyset$, stop; $\bar{x}$ is a quasistationary solution. Otherwise partition FN into $LFN \cup QFN$ (the sets of linear free nonbasic indices and quadratic free nonbasic indices), where

$$LFN = \{j \in FN | f_j \text{ is linear on the interval which includes } \bar{x}_j\},$$
$$QFN = FN\backslash LFN.$$

Similarly partition FB into $LFB \cup QFB$. Let T_{LL} be the submatrix of T_{FF} consisting of t_{ij} with $i \in LFB$ and $j \in LFN$. Similarly define T_{LQ}, T_{QL} and T_{QQ}. (See Figure 2.) Pivot on nonzero entries $t_{ij} \in T_{QL}$ until either $T_{QL} = \emptyset$ or $T_{QL} = 0$. Go to Step 3.

Step 3. If $LFN = \emptyset$, go to Step 4. Otherwise for each $j \in LFN$ calculate

$$\delta_j = f_j'(\bar{x}_j) + \sum_{i \in LFB} t_{ij} f_i'(\bar{x}_j).$$

Let $\bar{y}$ be the elementary vector corresponding to j. That is, its jth element is 1 and its basic variables are given by the jth column of T; all other components are zero. If $\delta_j < 0$, do one-piece line search with $(\bar{x}, \bar{y})$; if $\delta_j > 0$, do one-piece line search with $(\bar{x}, -\bar{y})$; if $\delta_j = 0$ and $\epsilon_0 \neq +\infty$, do one-piece line search with $(\bar{x}, \bar{y})$; if $\delta_j = 0$ and $\epsilon_0 = +\infty$, do one piece line search with $(\bar{x}, -\bar{y})$. Repeat Step 3 until either the one-piece line search indicates that inf(SP4)$=-\infty$ or until $LFN = \emptyset$. In the former case, stop; in the latter case(see Figure 2), go to Step 4.

Step 4. At this point, $LFN = \emptyset$. If $QFN = \emptyset$, stop; $\bar{x}$ is a quasistationary solution. Otherwise solve the following unconstrained quadratic program:

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{2} y_{FN} \cdot P_{FN} y_{FN} + q_{FN} \cdot y_{FN} + \frac{1}{2} y_{FB} \cdot P_{FB} y_{FB} + q_{FB} \cdot y_{FB} \\
\text{subject to} \quad & T_{FF} y_{FN} = y_{FB} \\
\text{where} \quad & y_{FN} = \{y_j | j \in FN\} \quad y_{FB} = \{y_i | i \in FB\} \\
& P_{FN} = \operatorname{diag}[f_j''(x_j)]_{j \in FN} \quad P_{FB} = \operatorname{diag}[f_i''(x_i)]_{i \in FB} \\
& q_{FN} = [f_j'(x_j)]_{j \in FN}^T \quad q_{FB} = [f_i'(x_i)]_{i \in FB}^T
\end{aligned}$$

Calculate $y_{FB} = T_{FF} y_{FN}$, then set

$$\bar{y} = [\bar{y}_1, \cdots, \bar{y}_k, \cdots, \bar{y}_n]^T, \quad \textit{where} \quad \bar{y}_k = \begin{cases} y_k & \text{if } k \in FB \cup FN, \\ 0 & \text{otherwise.} \end{cases}$$

Go to Step 5.

Step 5. Do a one-piece line search with (x^k, y^k). If the line search ends up with a point that is not on the relative boundary of the piece, stop; the point reached by the line search is a quasioptimal solution. If the line search shows the infimum of (PQP) is $-\infty$, stop; otherwise go to Step 1.

Figure 1. The Partition of the Simplex Tableaux

Figure 2. Further partition of the Simplex Tableaux

Justification of the algorithm

The algorithm starts with a feasible $\bar{x}$. After Step 1, we get the following simplex tableau:

$$\begin{pmatrix} x_{FB} \\ x_{IB} \end{pmatrix} = \begin{pmatrix} T_{FI} & T_{FF} \\ T_{II} & 0 \text{ or } \emptyset \end{pmatrix} \begin{pmatrix} x_{IN} \\ x_{FN} \end{pmatrix} + d.$$

Thus if $FN = \emptyset$, the values of all basic variables are uniquely determined by those of the imprisoned nonbasic variables. Then x is the unique feasible solution of (SP1), so it is quasistationary.

After Step 2 we get the simplex tableau

$$\begin{pmatrix} x_{LFB} \\ x_{QFB} \\ x_{IB} \end{pmatrix} = \begin{pmatrix} (\, T_{FI}\,) & \begin{pmatrix} T_{LQ} & T_{LL} \\ T_{QQ} & 0 \text{ or } \emptyset \end{pmatrix} \\ (\, T_{II}\,) & (\, 0 \text{ or } \emptyset\,) \end{pmatrix} \begin{pmatrix} x_{IN} \\ x_{QFN} \\ x_{LFN} \end{pmatrix} + d$$

Therefore in Step 3 we have $F'(\bar{x}, \pm\bar{y}) = \pm\delta_j$ (cf. Proposition(3.3.2)). Since only linear functions are involved in the one-piece line search procedures in Step 3, if $\epsilon_0 \neq +\infty$ and $\delta_j \neq 0$, the line search must finish at some relative boundary point of $[\tilde{c}^-, \tilde{c}^+]$ and thus decrease the size of $LFB \cup LFN$ by at least 1. In this case we go back to Step 3 to repeat the procedure. The iteration can happen only finitely many times before we go to Step 4. If $\epsilon_0 = +\infty$ and $\delta_j \neq 0$, we have $\inf(\text{PQP}) = \inf(\text{SP4}) = -\infty$ (cf. the statement of one-piece line search); if $\delta_j = 0$, then by the breakpoint convention, either along $\bar{y}$ or along $-\bar{y}$ we will find a relative boundary point. Therefore Step 3 will produce a feasible solution whose simplex tableau has $LFN = \emptyset$ and whose objective value is not greater than $F(\bar{x})$.

If $QFN = \emptyset$ in Step 4, then $FN = \emptyset$ and $\bar{x}$ is quas i station ary. Otherwise the simplex tableau will look like

$$\begin{pmatrix} x_{FB} \\ x_{IB} \end{pmatrix} = \begin{pmatrix} T_{FI} & T_{FF} \\ T_{II} & 0 \text{ or } \emptyset \end{pmatrix} \begin{pmatrix} x_{IN} \\ x_{FN} \end{pmatrix} + d,$$

where for each $j \in FN$, f_j is quadratic around $\bar{x}_j$. The one-piece line search along $\bar{y}$ in Step 5 will give the minimi z er of problem (SP1) if it does not stop at a relative boundary point of $[\tilde{c}^-, \tilde{c}^+]$, because $\bar{y}$ is the Newton direction for problem (SP1). If the line search ends at a relative boundary point of $[\tilde{c}^-, \tilde{c}^+]$, then by going back to Step 1, we will move at least one index from $FB \cup FN$ to $IB \cup IN$. Thu s after finitely many iterations, we will find that either the set FN is empty or line search along $\bar{y}$ does not stop at any relative boundary points of $[\tilde{c}^-, \tilde{c}^+]$. Both cases yield a quasistationary solution where corresponding objective value is not greater than the value at the starting point.

$(\mathcal{Q}.\mathcal{E}.\mathcal{D}.)$

3.3 Algorithm for Finding a Descent Vector

Now we turn to Step 3 of the model algorithm. The question is how to descend from a given feasible solution x. Obviously, either a one-piece or multi-piece line search in a descent direction will reach another feasible solution with lower objective value. Hence the problem reduces to getting a descent vector. The accurate definition of a descent vector is:

Definition (3.3.1). Suppose x is a feasible solution to (PQP), y satisfies $Ay = 0$ and $F'(x, y) < 0$. Then y is a *descent vector* at x.

Now suppose we have a simplex tableau $x_B = Tx_N + d$ on hand. For simplicity of notation, suppose $B = \{1, ..., m\}$ and $N = \{m+1, ..., n\}$. The j-th column of T, denoted by $[t_{1j}, ..., t_{mj}]^T$, corresponds to the nonbasic variable x_{m+j}, while the i-th row of T, denoted by $[t_{i1}, ..., t_{i,n-m}]$, corresponds to basic variable x_i.

Proposition (3.3.2). A necessary and sufficient condition for the elementary vector

$$[t_{1j}, ..., t_{mj}, 0, ..., 0, \underset{\substack{\uparrow \\ (m+j)-th\ component}}{1}, 0, ..., 0]^T$$

to be a descent vector is

$$\delta_j^+ = f_{m+j}^+(x_{m+j}) + \sum_{i=1}^{m} \max\{t_{ij} f_i^-(x_i), t_{ij} f_i^+(x_i)\} < 0.$$

Similarly, the elementary vector $[-t_{ij}, ..., -t_{mj}, 0, ..., 0, -1, 0, ..., 0]^T$ is a descent vector if and only if

$$\delta_j^- = f_{m+j}^-(x_{m+j}) + \sum_{i=1}^{m} \min\{t_{ij} f_i^-(x_i), t_{ij} f_i^+(x_i)\} > 0.$$

Proof. All the elementary vectors satisfy the system $Ay = 0$. Let the first elementary vector be z. Then by the definition of directional derivatives we have

$$\begin{aligned} F'(x,z) =& \lim_{t \downarrow 0} \frac{\sum_{j=1}^n f_j(x_j + tz_j) - \sum_{j=1}^n f_j(x_j)}{t} \\ =& \sum_{j=1}^n \lim_{t \downarrow 0} \frac{f_j(x_j + tz_j) - f_j(x_j)}{t} (\text{Note : No limit in this sum can be } -\infty.) \\ =& \sum_{j=1}^n f_j'(x_j, z_j) = \sum_{z_j > 0} z_j f_j^+(x_j) - \sum_{z_j < 0} z_j f_j^-(x_j) \\ =& \sum_{j=1}^n \max\{z_j f_j^-(x_j), z_j f_j^+(x_j)\} = \delta_j^+. \end{aligned}$$

The similar calculation gives

$$F'(x,-z) = f_{m+j}^-(x_{m+j}) + \sum_{i=1}^{m} \min\{t_{ij} f_i^+(x_i), t_{ij} f_i^-(x_i)\} = \delta_j^-.$$

The conclusion of the proposition follows.

($\mathcal{Q.E.D.}$)

According to Proposition (3.3.2), the determination of an elementary descent vector seems to be a quite simple issue. The complication comes from the so-called degeneracy. This will be discussed in the next section.

4. Degeneracy Processing

There is a problem left in Section 3. What if we have $\delta_j^- \leq 0 \leq \delta_j^+$ for all $j \in N$ in a given simplex tableau at a given point x? Can we find a descent direction by pivoting to another simplex tableau or can we declare the present x is an optimal solution?

To answer these questions, let us define the degeneracy of a simplex tableau at a given point x and look at the optimality condition of PQP, as stated in the following.

Definition (4.1). A simplex tableau is degenerate at x if there exists an index $i_0 \in B$ such that the subdifferential $\partial f_{i_0}(x_{i_0})$ is not a singleton.

In other words, if one of the basic variables x_{i_0} happens to be at a breakpoint of f_{i_0} and f_{i_0} is first order discontinuous at this point, then the corresponding simplex tableau is degenerate at x.

Proposition (4.2). Suppose that $x_B = TX_N + d$ is a simplex tableau of $Ax = b$. Then x is optimal to (PQP) if and only if there exist $v_i \in \partial f_i(x_i)$ for all $i \in B$ such that

$$-\sum_{i \in B} t_{ij} v_i \in \partial f_j(x_j) \ for \ all \ j \in N.$$

Proof. Since any vector (v_B, v_N) in the range space of A^T satisfies $v_N = -Tv_B$ and vice versa, the condition of the proposition is equivalent to that of the vector $v = (v_i, i \in B, -\sum_{i\in B} t_{ij}v_i, j \in N)$ which is in the range space of A^T and satisfies $v_k \in \partial f_k(x_k)$ for all k, or $v \in \partial F$. This means there exists a vector u such that $-A^T u \in \partial F(x)$, or $0 \in \partial F(x) + A^T u$, so the Kuhn-Tucker optimality condition is satisfied and x is an optimal solution to the (PQP) (see Theorem 28.3 of [5]).

$(\mathcal{Q}.\mathcal{E}.\mathcal{D}.)$

Corollary (4.3). If the simplex tableau is nondegenerate at x and $\delta_j^- \le 0 \le \delta_j^+$ for all $j \in N$, then x is optimal.

Proof. The nondegeneracy indicates that there exist $\{v_i\} = \partial f_i(x_i)$ for all $i \in B$. Now the condition $\delta_j^- \le 0 \le \delta_j^+$, for all $j \in N$ can be written as

$$f_j^-(x_j) + \sum_{i\in B} t_{ij}v_i \le 0 \le f_j^+(x_j) + \sum_{i\in B} t_{ij}v_i$$

for all $j \in N$. Hence the condition of Proposition (4.2) is satisfied.

$(\mathcal{Q}.\mathcal{E}.\mathcal{D}.)$

When the simplex tableau $x_B = Tx_N + d$ is degenerate at x, the following algorithm will determine if x is optimal. If x is not optimal, it then finds an elementary descent direction. Thus our model algorithm can go on even if degeneracy is present.

Algorithm (4.4). Start with a feasible solution x and a simplex tableau which is degenerate at x.
Step 0. Arbitrarily choose $v_i = f_i^-(x_i) > -\infty$ or $v_i = f_i^+(x_i) < +\infty$ for all $i \in B$. For the exceptional case that $f_i^-(x_i) = -\infty$ and $f_i^+(x_i) = +\infty$, variable x_i can be actually disregarded from the beginning of our consideration, so we assume this case does not arise.
Step 1. Check for all $j \in N$ if $\delta_j^- \le 0 \le \delta_j^+$. If not, then a descent direction is found; return to main algorithm. Otherwise go to Step 2.
Step 2. Check for all $j \in N$ if $-\sum_{i\in B} t_{ij}v_i \in \partial f_j(x_j)$. If yes, then x is optimal; stop. Otherwise choose the smallest index j_0 such that the above inclusion is not valid. This x_{j0} will be the entering variable. Go to Step 3.
Step 3. If

$$(*) \qquad f_{j0}^-(x_{j0}) + \sum_{i\in B} t_{ij}v_i > 0$$

in Step 2, then find the smallest index i_0 such that

$$f_{j_0}^+(x_{j_0}) + \sum_{i\in B, i<i_0} \max\{t_{ij}f_i^-(x_i), t_{ij}f_i^+(x_i)\} + \sum_{i\in B, i\ge i_0} v_i t_{ij_0} \le 0;$$

if

$$(**) \qquad f_{j0}^+(x_{j0}) + \sum_{i\in B} t_{ij}v_i < 0$$

in Step 2, then find the smallest index i_0 such that

$$f_{j_0}^- + \sum_{i\in B, i<i_0} \min\{t_{ij}f_i^-(x_i), t_{ij}f_i^+(x_i)\} + \sum_{i\in B, i\ge i_0} v_i t_{ij_0} \ge 0.$$

The variable x_{i_0} will be the leaving variable. Pivot on (i_0, j_0), set $v_{j_0} = f_{j_0}^-(x_{j_0})$ or $f_{j_0}^+(x_{j_0})$ depending on the former or latter case and go to Step 1.

Justification of the Algorithm
The descent direction in Step 1 and the optimality in Step 2 can be justified by Propositions (3.3.2) and (4.2), respectively. Hence what we need to show is the non-cycling property of the pivoting operation in Step 3.

Imagine our objective function f_j is replaced by a piecewise linear function:

$$\bar{f}_j(y_j) = \begin{cases} f_j(x_j) + f_j^+(x_j)(y_j - x_j) & \text{if } y_j \geq x_j; \\ f_j(x_j) + f_j^-(x_j)(y_j - x_j) & \text{if } y_j \leq x_j. \end{cases}$$

then the algorithm (4.4) will identify to a non-cyclic algorithm for the piecewise linear programming problem with the objective function $\sum \bar{f}_j$.(See section 5 of [3] for detail.) Hence cycling is impossible. Another thing that needs to be mentioned is the choice of v_{j_0} is always possible in Step 3 because the conditions (*) or (**) ensures the value $f_{j_0}^+(x_{j_0})$ or $f_{j_0}^-(x_{j_0})$ is finite when it is chosen.

$(\mathcal{Q}.\mathcal{E}.\mathcal{D}.)$

5. Computational Experiment

In this section we describe the computational program and the results of our numerical experiments.

5.1 Experiment Design and Program

Since there seems to be no algorithm previously reported in the literature for PQP problems that are not reformulated as quadratic programming problems, the primary goal of the experimental work has not been to compare the effectiveness of our algorithm with others. Instead, we have mostly been interested in the following comparison: If the numbers of variables and constraints are fixed, will the increase in the number of pieces of f_j substantially increase the computing time?

Let n, m and k_j be the number of variables, the number of constraints and the number of pieces of the j-th variable, respectively. We refer to n as the *dimension* of the problem. Since a good program package usually needs years to develop and should be the topic of future research, we do not think our program code would be superior in solving purely quadratic problems, especially when the dimension is high. For our purpose, it is sufficient just to test the problems of small dimension, so as to avoid possible numerical difficulties such as the accumulation of round-off errors and ill-conditioned linear equations. On the other hand, we need to drive the numbers k_j high enough to get meaningful information. For simplicity, in our test all the k_j's are equal, and $m = \lfloor \frac{n}{2} \rfloor$. Let k be the common number of pieces. Then the pair (n, k) completely specifies the size of a tested problem. Let $T(n, k)$ denote the CPU time.

Roughly speaking, our program consists of two parts: the calculation of quasistationary solutions and multi-piece line search (including the determination of optimality and otherwise a descent direction) as we described in Section 3. Within this framework we find that if a quasistationary solution is calculated after several multi-piece line searches instead of after every such search, then considerable computation time can be saved. This is likely because the procedure of computing a quasistationary solution, which includes finding several Newton directions and making several pivotings, needs much more time than the procedure of multi-piece line searches. In our tested program code, the number of multi-piece line searches between two consecutive quasistationary solutions is taken to be 5. In computing the quasistationary solutions we strictly follow the steps described on Algorithm(3.2.4), i.e.

$$\begin{aligned} &[\text{making } T_{IF} = 0 \text{ or } \emptyset] \rightarrow \quad [\text{making } T_{QL} = 0 \text{ or } \emptyset] \rightarrow \ [\text{making } LFN = \emptyset] \rightarrow \\ &[\text{calculating } M = P_{FN} + T_{FF}^T P_{FB} T_{FF}] \rightarrow \ [\text{calculating the Newton direction } y]. \end{aligned}$$

If for any reason a variable x_j is changed after one iteration, a subroutine named LOCATE determines the piece (or breakpoint) x_j where is located; then a subroutine SUBDIF is called to find $\partial f_j(x_j)$ for next iteration. Figure 3 is the flow chart of the program.

5.2 Tested Problems

Figure 3. The Flow Chart of the Program

All tested problems were generated randomly by a program called PQPGEN. We input the values of m, n and k. PQPGEN then generates k breakpoints and a characteristic curve for each x_j such that $0 \in \operatorname{dom} F$. (Recall that F denotes the objective function of PQP.) After this, PQPGEN produces an initial simplex tableau $x_B = Tx_N$, where $B = \{1, \cdots, m\}$ and $N = \{m+1, \cdots, n\}$. Since all entries in T come from a pseudo-random number subroutine, T is usually dense. Because all breakpoints are finite numbers, the generated problems always have optimal solutions.

The feasibility of the zero vector and the homogeneity of the Tucker representation are not restrictions, since a simple transformation will make any PQP (if feasible) satisfy these conditions. In contrast, the suppression of the initial step (finding a feasible so l ution) makes the experimental results more accurately reflect the computation time that we wish to know.

5.3 Computational Results

The computational experiments ware conducted on VAX750/ UNIX System. The numerical results are listed in Table 1.

Although the code is written in a nonprofessional way, all the tested problems were still solved in reasonable time. The iterative sequence approaches an optimal solution in the following pattern: The simplex descent step first moves the initial solution to a point closer to the minimum. The first multi-piece line search usually crosses fairly many pieces. Then the algorithm spends lots of time locating the first quasistationary solution. For example, if the number of variables is 100 and the number of constraints is 50, then Algorithm (3.2.4) (the subprogram of finding quasistationary solutions) starts usually with a feasible solution with 50 free nonbasic variables. This leads to solving a linear equation system with 50 unknowns. After this a one-piece line search, probably including a pivoting operation in a 50×100 matrix, is performed and typically ends up with a new feasible solution having 49 free nonbasic variables. This procedure is subsequently repeated up to 49 times, each time with a problem of dimension one less, until a quasistationary solution is obtained. The algorithm then needs much less time to make one iteration because most of the nonbasic variables stay imprisoned and consequently the linear equation systems encountered in the calculation of the Newton directions have very small dimension (1 or 2, likely). Finally, when the solution sequence comes to the pieces where the optimal solution resides, a reverse mode of operation is experienced: The line search along the Newton direction almost always ends up with a relative interior point of a certain piece. Then the simplex descent step moves more and more imprisoned nonbasic variables out of breakpoints, and therefore the linear equation systems solved for finding Newton directions get larger and larger until an optimal solution is reached. Another phenomenon observed is that the amount of pivoting is low, although pivoting theoretically can happen after each one-piece line search.

The computational results show no sharp increase in $T(n, k)$ when k increases drastically. In fact, the quantity $T(n, k)$ increases roughly linearly with k. One can see that

$$2T(n, k) \leq T(n, 10k) \leq 10T(n, k) \quad \text{for all tested } n, k.$$

On the other hand, this result appearently indicates that solving a piecewise quadratic problem is somewhat like solving a quadratic program of the same dimension. This point is supported by comparing all the tested pairs $T(n, k)$ with $T(n, 1)$. The computational efficiency so demonstrated should at least encourage more research in dual methods of quadratic programming because, as we mentioned before, the dual of a quadratic program is in general piecewise quadratic. Besides, the insensitivity of computational time versus the number of pieces is good news, since some practical models generate PQP problems with a large number of pieces [9].

5.4 Conclusions

We offer a framework of a class of finitely convergent algorithms for PQP problems. One of them is described in detail. It is an extension of simplex methods of convex programming and active set methods of quadratic programming. A computational experiment is conducted based on this algorithm and randomly generated dense problems. The results of the experiment show that solving such a PQP problem is not much harder

n	m	k	$T(n,k)(sec.)$	n	m	k	$T(n,k)(sec.)$
5	2	1	0.3	15	7	1	2.3
5	2	10	0.7	15	7	10	4.1
5	2	20	1.3	15	7	20	5.9
5	2	30	1.4	15	7	30	6.4
5	2	40	2.2	15	7	40	7.8
5	2	50	2.2	15	7	50	8.6
5	2	60	2.8	15	7	60	10.9
5	2	70	3.0	15	7	70	12.1
5	2	80	3.8	15	7	80	16.1
5	2	90	3.7	15	7	90	13.7
5	2	100	4.1	15	7	100	14.2
5	2	200	9.1	15	7	200	26.0
5	2	300	12.8	15	7	300	37.3
5	2	400	15.5	30	15	1	16.0
5	2	500	19.7	30	15	5	19.6
5	2	600	24.2	30	15	10	23.2
5	2	700	28.1	30	15	15	18.7
5	2	800	32.0	30	15	30	21.1
5	2	900	36.5	30	15	50	28.9
5	2	1000	39.0	30	15	70	32.3
10	5	1	0.9	30	15	100	49.3
10	5	10	1.9	30	15	150	59.6
10	5	20	2.6	50	25	1	81.1
10	5	30	4.9	50	25	10	79.9
10	5	40	5.5	50	25	20	68.5
10	5	50	5.6	50	25	30	71.3
10	5	60	6.7	50	25	40	83.7
10	5	70	7.6	50	25	50	131.4
10	5	80	7.4	50	25	60	100.2
10	5	90	9.0	50	25	70	99.5
10	5	100	9.2	50	25	80	80.9
10	5	200	17.4	50	25	90	94.1
10	5	300	24.6	50	25	100	99.9
10	5	400	33.7	100	50	1	734.2
10	5	500	43.1	100	50	10	1420.3

Table 1. The Computational Time $T(n,k)$

than solving a QP problem of the same number of variables. This, we wish, will stimulate more research in the theory, application and software of PQP.

References

[1]. R. Fletcher, *Practical Methods of Optimization,* Vol. 2, John Wi l ey and Sons, New York and Toronto (1980).

[2]. R. Fourer, "A simplex algorithm for piecewise linear programming I: derivation and proof," *Mathematical Programming* 33 (1985), 204-233.

[3]. R. Fourer, "A simplex algorithm for piecewise linear programming II:

finiteness, feasibility and degeneracy," *Technical Report 85-03, Department of Industrial Engineering and Management Sciences, Northwestern University* (1985).

[4]. A. Premoli, "Piecewise linear programming:the compact (CPLP) algorithm," *Mathematical Programming* 36 (1986) 210-227.

[5]. R. T. Rockafellar, *Convex Analysis,* Princeton University Press (1970).

[6]. R. T. Rockafellar, *Network Flow and Monotropic Optimization,* Jo h n Wiley and Sons, New York (1984).

[7]. R. T. Rockafellar, "Linear-quadratic programming and optimal control," *SIAM Journal of Control and Optimization* 25 (1987) 781-814.

[8]. R. T. Rockafellar and R. J.-B. Wets, "Linear-quadratic programming problems with stochastic penalties: the finite generation algorithm," in *Stochastic Optimization,* V. I. Arkin, A. Shiraev and R.J-B. Wets (eds.), Springer-Verlag Lecture Notes in Control and Information Sciences No. 81 (1987).

[9]. J. Sun, "Basic theories and selected applications of monotropic piecewise quadratic programming," *Technical Report 86-09, Department of Industrial Engineering and Management Sciences, Northwestern University* (1986).

[10]. W. Zangwill, *Nonlinear Programming—A Unified Approach,* Prentice Hall, Englewood Cliffs, NJ(1969).