The Branch and Bound Algorithm for Solving a Sort of Non-smooth Programming on Simplex

Zai-En Hou\(^1,\dagger\) \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \qua
solve the programming (see [1–3]). And some modern methods for solving smooth programming are applied to solve the programming, such as bundle method, trust region method, reformed Newton method and SQP method, which is a successive approximation method with quadratic programming (see [4–9]). Some of these methods have requests or restrictions on smoothness or convexity to the function in the programming. The typical example is the Lipschitz optimization in which the Lipschitzian continuity is assumed to the objective and constraint function (see [10,11]). Can these restrictions be weakened in model (NSP)? These motivate us to study the properties of the Hölder function, the approximation of the function by using Bernstein α-polynomial and the strategy for branching and bounding. Our goal is to develop a new method for solving the programming where the objective function and constrained functions are all Hölder continues (see [12]).

The content of this paper is as follows. In section 2 we review the properties of the Hölder function and the approximation of the function by using Bernstein α-polynomial. In section 3 we construct the algorithm and study the convergence of the algorithm for solving the programming. In section 4 we give a numerical example to illuminate the feasibility of the algorithm. We end the paper with the conclusion of the paper in section 5.

2 The Properties of Hölder Function

2.1 Hölder Function

In this section we review the concept of Hölder function and study several properties of the function.

Definition 1. Let \( f(x) \) be a real function on \( P \subset \mathbb{R}^s \), \( f(x) \) is called a Hölder function on \( P \) (or Hölder continuous) if there exist constant \( L = L(f,P) > 0 \) and \( \gamma > 0 \) such that

\[
|f(x_2) - f(x_1)| \leq L \|x_2 - x_1\|_\gamma, \text{ for all } x_1, x_2 \in P.
\] (1)

Where constants \( L = L(f,P) > 0 \) and \( \gamma > 0 \) are called Hölder constants of \( f(x) \).

In the definition the norm \( \| \cdot \|_\gamma \) is the general Euclidean norm. In practice, the following \( l_p \)-norm is often adopted

\[
\|x\|_p = \left( \sum_{i=1}^{s} |x_i|^p \right)^{\frac{1}{p}}, \quad (1 \leq p \leq \infty),
\] (2)

where \( \|x\|_\infty = \max_{i=1,\cdots,s} |x_i| \).

Obviously, Hölder function \( f(x) \) is a Lipschitz function on \( P \) when \( \gamma = 1 \), that is, Lipschitz function is a special kind of Hölder function. Even so, the continuity of them is alike, and other properties of them are analogous as well.

Lemma 1. Let \( f(x), g(x) \) and \( h(x) \) be Hölder functions on the compact set \( P \subset \mathbb{R}^s \), then for any \( x \in P \), there exists a neighborhood \( U(x) \) at \( x \), such that
(i) Every linear combination of $f_i(x) (i = 1, \ldots, m)$ is a Hölder function on $U(x) \cap P$;
(ii) $\max_{i=1,\ldots,m} f_i(x)$ and $\min_{i=1,\ldots,m} f_i(x)$ are Hölder functions on $U(x) \cap P$;
(iii) $f(x) \cdot g(x)$ is a Hölder function on $U(x) \cap P$.

Proof. The constructive method may be used in the proofs of (i) $\sim$ (iii). The proof of $\max_{i=1,\ldots,m} f_i(x)$ in (ii) is given here, the others are similar.

Suppose $L_i$ and $\gamma_i (i = 1, \ldots, m)$ are Hölder constants of function $f_i(x)$ $(i = 1, \ldots, m)$, respectively, by the definition 1, we have following inequalities

$$|f_i(x_2) - f_i(x_1)| \leq L_i \|x_2 - x_1\|^p, \text{ for all } x_1, x_2 \in P, (i = 1, \ldots, m). \quad (3)$$

Let

$$F(x) = \max_{i=1,\ldots,m} f_i(x) \quad (4)$$

and $U(x) = \{\|x - x\| < d\}$, for any $x \in P$, we choose $d$ such that $0 < d \leq 1$, then for all $x_1, x_2 \in U(x) \cap P$, we have following inequalities

$$|F(x_2) - F(x_1)| = |\max_i f_i(x_2) - \max_i f_i(x_1)|$$

$$\leq \max_i |f_i(x_2) - f_i(x_1)|$$

$$\leq \max_i \{L_i \|x_2 - x_1\|^p\} \quad (5)$$

Further, let $L = \max_i L_i$, $\gamma = \min_i \gamma_i$, the inequality (5) can be denoted as follows

$$|F(x_2) - F(x_1)| \leq L \|x_2 - x_1\|^p. \quad (6)$$

This shows that $\max_{i=1,\ldots,m} f_i(x)$ is a Hölder function on $U(x) \cap P$. 

Lemma 2. Let $f_i(x)$ $(i = 1, \ldots, m)$ be a Hölder function on the compact set $P \subset \mathbb{R}^d$ and $L_i, \gamma_i (i = 1, \ldots, m)$ be the Hölder constants respectively, then for any $x \in P$, there exists a neighborhood $U(x)$ at $x$, such that for any $p$ $(1 \leq p \leq \infty)$ the function $f(x) = (\sum_{i=1}^m |f_i(x)|^p)^{\frac{1}{p}}$ is a Hölder function on $U(x) \cap P$ and the constants $L = \sum_{i=1}^m L_i$ and $\gamma = \min_i \gamma_i$ are its Hölder constants.

Proof. Let $F(x) = (f_1(x), \ldots, f_m(x))^T$, then $f(x)$ is the $l_p$-norm of $F(x)$, that is $f(x) = \|F(x)\|_p$. Let $U(x) = \{\|x - x\| < d\}$, for any $x \in P$, where we choose $d$
such that $0 < d \leq 1$, then for all $x_1, x_2 \in U(x) \cap P$, we have following inequalities
\[
||F(x_2)||_p - ||F(x_1)||_p \leq ||F(x_2) - F(x_1)||_1
\]
\[
= \sum_{i=1}^m |f_i(x_2) - f_i(x_1)|
\]
\[
\leq \sum_{i=1}^m \|x_2 - x_1\|_p^{\gamma_i}
\]
\[
\leq \sum_{i=1}^m L_i \|x_2 - x_1\|^{\min\{\gamma_i\}}
\]
\[
= \left( \sum_{i=1}^m L_i \right) \|x_2 - x_1\|^{\min\{\gamma_i\}}
\] (7)

Therefore, let $L = \sum_{i=1}^m L_i$, $\gamma = \min\{\gamma_i\}$, the inequality (7) can be expressed as follows
\[
|f(x_2) - f(x_1)| \leq L \|x_2 - x_1\|^\gamma.
\] (8)

It is shown that $f(x)$ is a Hölder function on $U(x) \cap P$. □

Actually, the functions in the results of the lemma are all local Hölder function.

2.2 The Approximation of Hölder Function

From the definition 1, Hölder function is continuous on $P$ while $P(\subset \mathbb{R}^s)$ is simplex (viz. hyper-simplex). In this case the Hölder function can be approximated by Bernstein $\alpha$- polynomials (like in paper [13]). But the labor of calculation of the approximation is increasing rapidly when the degree of the Bernstein $\alpha$- polynomials is increasing in higher order. The higher degree Bernstein $\alpha$- polynomials can not be used in practical computation. We pay attention to the process of branching and bounding, in which the region is divided smaller and smaller. Can the approximation by Bernstein $\alpha$- polynomials be applied in this process, that is, can the heightening of the degree of the polynomials be substituted by the dwindling of the region? If it is possible, we can conclude that the Hölder function can be approximated by Bernstein $\alpha$- polynomials in the process mentioned. We explain the feasibility of this approximation in the section below, then we will give the application of the approximation to the approximation algorithm in the next section.

**Lemma 3.** Let $f(x)$ be a Hölder function on the simplex $P \subset \mathbb{R}^s$, $\{P_m\}$ be a sequence of simplexes such that
\[
P = P_0 \supset P_1 \supset \cdots \supset P_m \supset \cdots,
\]
and $d_m = \sup\{\|x_2 - x_1\| | x_2, x_1 \in P_m\}$ be the diameter of simplex $P_m$ ($m = 0, 1, 2, \cdots$), such that $d_m \to 0$, as $m \to \infty$. And let
\[
B_m^n(f, x, \alpha) = \sum_{|k| \leq n} f((\frac{k}{n})^\alpha)B_n^k(x, \alpha)
\] (9)
be the Bernstein $\alpha$-polynomials of $f(x)$ on $P_m (m = 0, 1, 2, \cdots, \cdots)$, then we have the following limit
\[
\lim_{m \to \infty} (B_m^n (f, x, \alpha) - f(x)) = 0.
\] (10)

**Proof.** It is easy to know the $f(x)$ is a Hölder function on the simplex $P_m \subset P (m = 0, 1, \cdots)$ from the assumption of the lemma. Therefore, we have following inequality:
\[
|f(x_2) - f(x_1)| \leq L ||x_2 - x_1||^\gamma, \quad x_2, x_1 \in P_m (m = 0, 1, \cdots)
\] (11)
where $L$ and $\gamma > 0$ are the Hölder constants. We also have inequalities
\[
|B_m^n (f, x, \alpha) - f(x)| = \left| \sum_{|n| \leq n} \left( f \left( \frac{k}{n} \right)^\frac{j}{\alpha} - f(x) \right) B_{n,k} (x, \alpha) \right|
\leq \sum_{|n| \leq n} \left| f \left( \frac{k}{n} \right)^\frac{j}{\alpha} - f(x) \right| B_{n,k} (x, \alpha)
\leq \sum_{|n| \leq n} L \left\| f \left( \frac{k}{n} \right)^\frac{j}{\alpha} - x \right\|^\gamma B_{n,k} (x, \alpha)
\leq Ld_m^\gamma \sum_{|n| \leq n} B_{n,k} (x, \alpha) = Ld_m^\gamma.
\] (12)

From the assumption, when $m \to \infty$, $d_m \to 0$. Because of this, for any $\varepsilon > 0$, there exists a positive number $N$, such that $d_m < \left( \frac{\varepsilon}{L} \right)^{\frac{1}{\gamma}}$ as $m > N$. Thus
\[
|B_m^n (f, x, \alpha) - f(x)| < L \left( \frac{\varepsilon}{L} \right)^{\frac{1}{\gamma}} = \varepsilon.
\] (13)
It shows the limit (10) holds. \hfill $\square$

### 2.3 The Non-smooth Programming

For the non-smooth programming
\[
(\text{NSPH}) \begin{cases} 
\min & f(x) \\
\text{s.t.} & g_i(x) \leq 0, (i = 1, 2, \cdots, m) \\
& h_j(x) = 0, (j = 1, 2, \cdots, n),
\end{cases}
\]
where $f(x), g_i(x), h_j(x) (i = 1, 2, \cdots, m; j = 1, 2, \cdots, n)$ are Hölder functions, we define a Lagrange penalty function as follows
\[
L(x, M) = f(x) + M \left( \sum_{i=1}^m \max \{g_i(x), 0\} + \sum_{j=1}^n h_j^2(x) \right),
\]
where $M$ is a large positive number. Then we choose a huge simplex $D$ which comprise the feasible region of the programming (NSPH). The programming can be solved by solving the problem

$$\begin{align*}
\min_{x} & \quad L(x, M) \\
\text{s.t.} & \quad x \in D.
\end{align*}$$

Therefore, only do we need to solve the problem

$$\begin{align*}
\min_{x} & \quad f(x) \\
\text{s.t.} & \quad x \in D, \quad (14)
\end{align*}$$

or the problem

$$\begin{align*}
\min_{x} & \quad B_{n}(f, x, \alpha) \\
\text{s.t.} & \quad x \in D, \quad (15)
\end{align*}$$

where the $L(x, M)$ is denoted as the new function $f(x)$ in problem (14) and (15).

3 The Algorithm and Its Convergence

The general branch and bound methodology is applicable to broad classes of optimization problems. The branch and bound algorithms are based upon operations of partition, sampling, and subsequent lower and upper bounding procedures, in which these operations are applied iteratively to the collection of active (‘candidate’) subsets within the feasible set. In the section we describe the algorithm for solving the non-smooth programming ((NSPH) or (14)), in which we combine the strategy of branch and bound with the approximation of Hölder function by Bernstein $\alpha$-polynomials. Moreover, we study the convergence of the algorithm.

3.1 The Basic Algorithm

Compared with general branch and bound methodology, here we approximate the function $f(x)$ with its Bernstein $\alpha$-polynomials $B_{n}(f, x, \alpha)$ and we have the inequality

$$\min_{x \in D} f(x) \leq \min_{x \in \Omega} B_{n}(f, x, \alpha), \quad (16)$$

our method did not need the process of bounding in formally, it only needs the process of branching and pruning. Thus, rules of branching and pruning is the key of the method. According to these rules, we can preserve the solution in the sequence of the simplex, and we need solve lesser subproblem as well. Thus the labor of calculation needn’t be added a great deal.

The Branching Rule. To guarantee the convergence of the algorithm, we partition the simplex $P_{k}$ with bisection method, that is, we bisect the simplex $P_{k}$ at the midpoint of its longest edge. Where $P_{k}$ is partitioned as two simplex: $P_{k1}, P_{k2}$. The diameter of simplex $P_{k}(i = 1, 2)$ is dwindled step by step with the process of the partition continuing. Moreover, the diameter of the simplex $P_{k}(i = 1, 2)$ converges to zero as $k \to \infty$. 
The Pruning Rule. To set up the pruning rule, we need to determine the initial threshold value $\mu_0$ and the degree $n$ of the polynomials.

In theory, we can choose the initial threshold value $\mu_0 > 0$ and the degree $n$, such that

$$|B_n(f, x, \alpha) - f(x)| \leq \mu_0, \ x \in P_0$$

(17)

where $P_0 = D$, since we have the following limit

$$\lim_{n \to \infty} B_n(f, x, \alpha) = f(x), \ x \in P_0.$$  

(18)

But the degree $n$ may be too large to be used in actual computation.

In practical computation, we can first choose proper $n$ according to the properties of the function, where a smaller $n$ is chosen generally. Then we determine $\mu_0 > 0$ with the bisection method or 0.618 method (golden section method). Moreover, the following inequalities hold,

$$\min_{x \in P_0} B_n(f, x, \alpha) - \mu_0 \leq \min_{x \in P_0} f(x) \leq \min_{x \in P_0} B_n(f, x, \alpha).$$

(19)

Let $u = \min_{x \in P_0} B_n(f, x, \alpha)$ be the upper bound of the optimization value of objective function, according to the branching rule, we partition the simplex $P_0$ as: $P_0^1, P_0^2$, and solve $\min_{x \in P_0^1} B_n(f, x, \alpha), \min_{x \in P_0^2} B_n(f, x, \alpha)$ at the same time. Then comparing the optimization values of them with $u$, if $u > \min_{x \in P_0} B_n(f, x, \alpha)$, we update the upper bound $u$ as $u = \min_{x \in P_0} B_n(f, x, \alpha)$. Actually, that can be expressed as follows,

$$u = \min_{x \in P_0} \left\{ \min_{x \in P_0^1} B_n(f, x, \alpha), \min_{x \in P_0^2} B_n(f, x, \alpha) \right\} \leq \min_{x \in P_0} B_n(f, x, \alpha),$$

where the Bernstein $\alpha$-polynomial $B_n(f, x, \alpha)$ in $\min_{x \in P_0^1} B_n(f, x, \alpha), \min_{x \in P_0^2} B_n(f, x, \alpha)$ and $\min_{x \in P_0} B_n(f, x, \alpha)$ is the Bernstein $\alpha$-polynomial $B_n(f, x, \alpha)$ on corresponding simplex, respectively. For convenience, we denote them as the same front though they are different. This is the same in lemma 4 and hereinafter.

Like the proof of the theorem 3 in paper [13], we have following inequality

$$|B_n(f, x, \alpha) - f(x)| < \frac{M}{n}.$$
Afterwards, we do the process of branch and bound on $P_1$, and gain the new set of reserved branches $P_2$. Generally, repeating this process, we can obtain the set of reserved branches

$$P_m = \{P_{m1}, P_{m2}, \cdots, P_{ml_m}\}, \quad (20)$$

and corresponding upper bound $u$, threshold value $\mu_m$, diameter $d_m$.

Until satisfactory precision being reached, the above process is repeated.

The Terminate Rule. For given precision $\epsilon > 0$, if

$$\mu_m < \epsilon \text{ or } d_m < \epsilon,$$

then stop the process, and solve the optimization solution and value.

**Algorithm 1. (Basic Algorithm)**

**Step 0:** Let an accuracy $\epsilon > 0$ be given. Initialization: Initialize iterative number $m := 0$, initialize the set of reserved branches $P_0 = \{P_{01}\}$, $P_{01} = D$, choose proper integer $n$, and solve

$$\min_{x \in P_{01}} B_n(f, x, \alpha)$$

to obtain the solution $x_0$ and initial upper bound

$$u_0 = \min_{x \in P_{01}} B_n(f, x, \alpha),$$

and determine the initial threshold value $\mu_0$. Let $d_0$ be the diameter of $P_{01}$, which is called as diameter of $P_0$ too.

**Step 1:** If $\mu_m < \epsilon$ or $d_m < \epsilon$, then stop the iterative process. We obtain $u_m$ is the optimization value and the corresponding solution is the optimization solution. Otherwise, go to step 2.

**Step 2:** Denote the set of reserved branches as:

$$P_m = \{P_{m1}, P_{m2}, \cdots, P_{ml_m}\}. \quad (21)$$

Bisect every $P_{mj}(1 \leq j \leq l)$ as: $P_{mj}^1, P_{mj}^2(1 \leq j \leq l_m)$.

**Step 3:** Solve problems

$$\min_{x \in P_{mj}} B_n(f, x, \alpha), \quad j = 1, 2, \cdots, l_m; \ i = 1, 2.$$

Get the solutions $x_{mj}^i(j = 1, 2, \cdots, l_m; \ i = 1, 2)$; determine the upper bound

$$u_{m+1} = \min_{1 \leq j \leq l_m, i = 1, 2} \left\{ \min_{x \in P_{mj}} B_n(f, x, \alpha) \right\},$$

and corresponding solution $x_{m+1}$. 

---

Solving a Sort of Non-smooth Programming on Simplex

277
Step 4: Let $\mu_{m+1} = \frac{\mu_m}{2}$ be the new threshold value, if
\[
\min_{x \in P_{m+1}} B_n(f, x, \alpha) > u + \mu_{m+1},
\]
we abnegate the corresponding branch $P_{m+1}$. Otherwise, we save the corresponding branch, which constitute the new set of reserved branches
\[
P_{m+1} = \{P_{(m+1)1}, P_{(m+1)2}, \cdots , P_{(m+1)l_{m+1}}\}.
\]
(22)

Let
\[
d_{m+1} = \max_{1 \leq j \leq l_{m+1}} \{d_{(m+1)j}\} \text{ is the diameter of } P_{(m+1)j}, \ 1 \leq j \leq l_{m+1}
\]
be the diameter of $P_{m+1}$.

Step 5: $m + 1 \Rightarrow m$, go to step 1.

3.2 The Convergence of the Algorithm

To give the convergence of the algorithm we review the following concept.

Definition 2. Let $P$ be a simplex, then partition the simplex. If this partition produce a sequence $\{P_m\}$ of partition set of the simplex $P$ such that
\[
P = P_0 \supset P_1 \supset \cdots \supset P_m \supset \cdots,
\]
and $\lim_{m \rightarrow \infty} d(P_m) = 0$. \(\lim_{m \rightarrow \infty} P_m = \bigcap_{m=0}^{\infty} P_m = \{\hat{P}\}\), where $d(P_m)$ is the diameter of $P_m$ and $\hat{P}$ is a point in the simplex $P$, then this partition $\{P_m\}$ of $P$ is called as exhaustive.

In the aforesaid algorithm, the set of partition is the partition of simplex, this simplex partition is exhaustive from the reference [14]. Therefore, the branch and bound algorithm is exhaustive. So the algorithm is convergent from the exhaustive of the algorithm and lemma (3).

Theorem 4. If the process of branch and bound algorithm can be done infinitely, and this process produce every sequence of the partition sets, such that
\[
P_{m_1} \supset P_{m_2} \supset \cdots \supset P_{m_k} \supset \cdots,
\]
and it is exhaustive, then
\[
u = \lim_{k \rightarrow \infty} u_{m_k} = \lim_{k \rightarrow \infty} f(x_{m_k})
\]
and the accumulation point $x^*$ of the sequence $\{x_m\}$ is the optimization solution of the programming (14).

Proof. There exists a accumulation point of the sequence $\{x_m\}$ since the set $D$ is a compact set in the assumption of the theorem. Let $x^*$ be the accumulation point, then
there exists a subsequence \( \{x_{m_k}\} \) of sequence \( \{x_m\} \) such that the subsequence \( \{x_{m_k}\} \) converge to the point \( x^* \) and corresponding sequence \( \{P_{m_k}\} \)

\[
P_{m_1} \supset P_{m_2} \supset \cdots \supset P_{m_k} \supset \cdots
\]
is exhaustive. Combining the continuity of the function \( f(x) \) and the results of lemma 4, we have the limit

\[
\lim_{k \to \infty} f(x_{m_k}) = f(x^*),
\]

and the limit

\[
\lim_{k \to \infty} u_{m_k} = \lim_{k \to \infty} f(x_{m_k}).
\]

Thus from the meaning of \( u_{m_k} \), we know the point \( x^* \) is the optimization solution of the programming (14) and the corresponding limit \( u = \lim_{k \to \infty} u_{m_k} \) is the optimization value.

Clearly, for given \( \varepsilon > 0 \), the process of branch and bound will be stopped at finite number of steps under the terminate rule and results of the theorem. In other words, the algorithm possesses the property of finite termination.

4 Numerical Example

Example 1. Solve the problem

\[
\max_{x \in S} f(x)
\]  

(23)

where \( S = \{x : x = (x_1, x_2, x_3)^T \in \mathbb{R}^3, 0 \leq x_i \leq 1, i = 1, 2, 3, x_1 + x_2 + x_3 \leq 1\} \), and

\[
\begin{align*}
f(x) &= \max \{f_1(x), f_2(x)\}, \\
f_1(x) &= -1.0 + 8x_1 + 8x_2 - 32x_1x_2, \\
f_2(x) &= 3.6 - 12x_1 - 4x_3 + 4x_1x_3 + 10x_1^2 + 2x_3^2.
\end{align*}
\]

This is a maximum-minimum problem, in which the maximum of two functions or several functions is no longer smooth. Otherwise, if the primary two functions or several functions are Hölder functions, the obtained function also is a Hölder function. This property of Hölder function is the same as that of Lipschitz function.

As mentioned before, the larger the integer \( n \) is, the larger the labor of computation Bernstein \( \alpha \)-polynomial is. So when we solve the problem

\[
\max_{x \in S} B_n(f, x, \alpha),
\]  

(24)

with the above algorithm, we choose \( n = 3 \) and \( \alpha = (1.5, 1, 1.25)^T \).

For convenience in computation, we choose a rectangle \( \mathcal{S} = \{x : x = (x_1, x_2, x_3)^T \in \mathbb{R}^3, 0 \leq x_i \leq 1, i = 1, 2, 3\} \supset S \) and solve the problem \( \max_{x \in \mathcal{S}} f(x) \), that is, we solve the problem \( \max_{x \in \mathcal{S}} B_n(f, x, \alpha) \) with the algorithm in which the rectangle \( \mathcal{S} \) substitute
<table>
<thead>
<tr>
<th>Sequence number</th>
<th>Rectangle $S_m$</th>
<th>Extreme point $x^m$</th>
<th>$\min B_m$</th>
<th>$f(x^m)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>[0, 1]</td>
<td>(.79622; 96929; 00000)</td>
<td>.93130</td>
<td>.3850</td>
</tr>
<tr>
<td>1</td>
<td>[0, .5] [0, 1]</td>
<td>(.49438; 100000; 37119)</td>
<td>-.102800</td>
<td>-.3636</td>
</tr>
<tr>
<td>2</td>
<td>.5 [0, 1]</td>
<td>(.50000; 99999; 50000)</td>
<td>-.101280</td>
<td>-.4000</td>
</tr>
<tr>
<td>3</td>
<td>[0, .5] [0, 1]</td>
<td>(.50000; 50000; 28167)</td>
<td>-.00794</td>
<td>1.0000</td>
</tr>
<tr>
<td>4</td>
<td>[0, .5] [0, 1]</td>
<td>(.49438; 100000; 37119)</td>
<td>-.10280</td>
<td>-.3636</td>
</tr>
<tr>
<td>5</td>
<td>[0, .5] [0, 1]</td>
<td>(.50000; 50000; 28167)</td>
<td>-.00790</td>
<td>1.0000</td>
</tr>
<tr>
<td>6</td>
<td>[0, .5] [0, 1]</td>
<td>(.50000; 100000; 35931)</td>
<td>-.10128</td>
<td>-.3640</td>
</tr>
<tr>
<td>7</td>
<td>[0, .5] [0, 1]</td>
<td>(.49070; 100000; 38142)</td>
<td>-.22548</td>
<td>-.3666</td>
</tr>
<tr>
<td>8</td>
<td>[0, .5] [0, 1]</td>
<td>(.48529; 100000; 50000)</td>
<td>-.19528</td>
<td>-.3978</td>
</tr>
<tr>
<td>9</td>
<td>[0, .5] [0, 1]</td>
<td>(.50000; 100000; 36889)</td>
<td>-.22062</td>
<td>-.3656</td>
</tr>
<tr>
<td>10</td>
<td>[0, .5] [0, 1]</td>
<td>(.50000; 50000; 100000)</td>
<td>-.18140</td>
<td>1.0000</td>
</tr>
<tr>
<td>11</td>
<td>[0, .25] [0, 1]</td>
<td>(.2500; 100000; 50000)</td>
<td>-.31322</td>
<td>3.0000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>18</td>
<td>[.75, 1] [0, 1]</td>
<td>(.7500; 100000; 50000)</td>
<td>.76724</td>
<td>.2250</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>28</td>
<td>[.5, .75] [.75, 1]</td>
<td>(.5000; 100000; 50000)</td>
<td>-.18140</td>
<td>-.4000</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 1: Results of example 1

In our approach, we give a algorithm for solving the nonsmooth programming. Except the usual strategy of branch and bound, we mainly utilize on the approximation properties of Hölder function by Bernstein $\alpha$- polynomial. The dwindling of the region substitute the heightening of the degree of the polynomials in the approximation. This technique largely cut down on the labor of the computation. The

5 Conclusion

In our approach, we give a algorithm for solving the nonsmooth programming.
convergence of the algorithm is guaranteed in theory and the feasibility of algorithm is validated from the example. At the end of the paper, we point out there are two difficulties while solving a problem with the algorithm, one is the choosing of the parameter $\alpha$ in polynomial $B_n(f, x, \alpha)$, another is the solving of the subproblems $\min_{x \in P_m} B_n(f, x, \alpha)$, though we solve it as a geometric programming [15,16]. Therefore, this study will be continued.

References


