Greedy Algorithm
for Sparse Monotone Regression*

Alexey R. Faizliev[0000–0001–6442–4361],
Alexander A. Gudkov[0000–0002–6472–4855],
Sergei V. Mironov[0000–0003–3699–5006], and Mikhail A. Levshunov
Saratov State University, Saratov, Russia
1 sidorovsp@info.sgu.ru

Abstract. The problem of constructing the best fitted monotone regression is NP-hard problem and can be formulated in the form of a convex programming problem with linear constraints. The paper proposes a simple greedy algorithm for finding a sparse monotone regression using Frank–Wolfe-type approach. A software package for this problem is developed and implemented in R and C++. The proposed method is compared with the well-known pool-adjacent-violators algorithm (PAVA) using simulated data.

Keywords: greedy algorithms, pool–adjacent–violators algorithm, isotonic regression, monotone regression

1 Introduction

The recent years have seen an increasing interest in shape-constrained estimation in statistics [10]. One of such problems is the problem of constructing monotone regression. The problem is to find best fitted non-decreasing points to a given set of points on the plane. The survey of results on monotone regression can be found in the book by Robertson and Dykstra [25]. The papers of Barlow and Brunk [3], Dykstra [16], Best and Chakravarti [4], Best [5] consider the problem of finding monotone regression in quadratic and convex programming frameworks.

Using mathematical programming approach the works [1, 21, 31] have recently provided some new results on the topic. The papers [7, 17] extend the problem to particular orders defined by the variables of a multiple regression. The paper [8] investigates a dual active-set algorithm for regularized monotonic regression.

Monotone regression is widely used in mathematical statistics [2, 10]; in smoothing of empirical data [15]; in shape-preserving approximation [19], [26], [30], [6], [27], [13]; in shape-preserving dynamic programming [9].

Constructing monotone regression we assume a relationship between a predictor \( x = (x_1, \ldots, x_n) \) and a response \( y = (y_1, \ldots, y_n) \). In the general case it is expected that \( x_{i+1} - x_i \neq \text{const}, x_i < x_{i+1}, i = 1, \ldots, n - 1 \).

* The work was supported by RFBR (grant 16-01-00507).
A sequence \( z = (z_1, \ldots, z_n) \in \mathbb{R}^n \) is called monotone if

\[ z_i - z_{i-1} \geq 0, \quad i = 2, \ldots, n. \]

Denote \( \Delta^n_1 \) the set of all vectors from \( \mathbb{R}^n \), which are monotone.

The problem of constructing monotone regression can be formulated in the form of a convex programming problem with linear constraints as follows: it is necessary to find a vector \( z \in \mathbb{R}^n \) with the lowest mean square error of approximation to the given vector \( y \in \mathbb{R}^n \) under condition of monotonicity of \( z \):

\[
f(z) = \frac{1}{n} \sum_{i=1}^{n} (z_i - y_i)^2 \rightarrow \min_{z \in \Delta^n_1}, \tag{1}
\]

In many situations researchers have no information regarding the mathematical specification of the true regression function. Typically, this involves non-decreasing of \( y_i \)’s with the ordered \( x_i \)’s. Such a situation is called isotonic regression. Isotonic regression (monotone regression) is a special case to the \( k \)-monotone regression [24].

It is well-known that the problem (1) is NP-hard problem [24]. In this paper we present a simple greedy algorithm which employs Frank–Wolfe-type approach for finding sparse monotone regression. A software package for this problem is developed and implemented in R and C++.

For the convenience of solving the problem (1), we move from points \( z_i \) to its increments \( \zeta_i \), where \( \zeta_i = z_{i+1} - z_i, \quad i = 1, \ldots, n-1, \quad \zeta_0 = z_1 \). Then monotonicity of \( z \) corresponds non-negativity of \( \zeta_i \)’s (except \( \zeta_0 \)). The proposed method is compared with the well-known pool-adjacent-violators algorithm (PAVA) using simulated data.

2 Algorithms for monotone regression

2.1 PAVA

Simple iterative algorithm for solving the problem (1) is called Pool-Adjacent-Violators Algorithm (PAVA) [11,24]. The work [4] examined the generalization of this algorithm. The paper [32] studied this problem as the problem of identifying the active set and proposed a direct algorithm of the same complexity as the PAVA (the dual algorithm).

PAVA computes a non-decreasing sequence of values \( z = (z_i)_{i=1}^n \) such that the problem (1) is optimized. In the simple monotone regression case we have the measurement pairs \( (x_i, y_i) \). Let us assume that these pairs are ordered with respect to the predictors. The following (Algorithm 1) is a pseudocode of PAVA for the problem. The generalized pool-adjacent-violators algorithm (GPAPA), which is a strict generalization of PAVA, was developed in the article [33].

The block values are expanded with respect to the observations \( i = 1, \ldots, n \) such that the final result is the vector \( z \) of length \( n \) with elements \( \bar{z}_i \) of increasing order [24].
Algorithm 1: **Pool-Adjacent-Violators Algorithm (PAVA)**

\begin{algorithm}
\begin{algorithmic}
\begin{itemize}
\item Let $z^{(0)}_j := y_i$ be the start point, $l = 0$;
\item The index for the blocks is $r = 1, \ldots, B$ where at step $l = 0$ we set $B := n$,
\hspace{0.5cm} i.e. each observation $z^{(0)}_r$ forms a block;
\end{itemize}
\Repeat
\begin{itemize}
\item (Adjacent pooling) Merge values of $z^{(l)}_r$ into blocks if $z^{(l)}_{r+1} < z^{(l)}_r$;
\item Solve $f(z)$ for each block $r$, i.e., compute the update based on the
\hspace{0.5cm} solver which gives $z^{(l+1)}_r := s(z^{(l)}_r)$, the solver $s$ is conditional
\hspace{0.5cm} (weighted) mean and (weighted) quantities;
\item If $z^{(l)}_{r+1} \leq z^{(l)}_r$ then $l := l + 1$;
\end{itemize}
\Until the $z$-blocks are increasing, i.e. $z^{(l)}_{r+1} \geq z^{(l)}_r$ for all $r$;
\item Return $z$;
\end{algorithmic}
\end{algorithm}

2.2 Frank–Wolfe type greedy algorithm

Frank–Wolfe method (or conditional gradient method) solves conditional convex optimization problems in vector finite-dimensional space. The method was introduced in 1956. The original algorithm did not use a fixed step size, and has the complexity of the linear programming. Frank–Wolfe method was developed by Levitin and Polyak in 1966, and V.F. Demianov and A.M. Rubinov generalized it to the case of arbitrary Banach spaces in 1970 [14]. Recently Frank–Wolfe type methods have caused an increased interest related to the possibility of obtaining sparse solutions, as well as a good scaling [12, 23]. In particular, [22, 34] researched algorithms for solving problems with penalty functions (instead of considering the conditional optimization problems). Besides, the paper [34] uses interlacing boosting with fixed-rank local optimization.

As it was mentioned above, for computational convenience of the problem (1), we moved from points $z_i$ to increments $\zeta_i = z_{i+1} - z_i$, $i = 1, \ldots, n - 1$, $\zeta_0 = z_1$. Then the problem (1) can be rewritten as follows:

$$g(\zeta) := \frac{1}{n} \sum_{i=1}^{n} \left( \sum_{j=0}^{i-1} \zeta_j - y_i \right)^2 \rightarrow \min_{\zeta \in S},$$

(2)

where $S$ denotes the set of all $\zeta = (\zeta_0, \zeta_1, \ldots, \zeta_{n-1}) \in \mathbb{R}^n$ such that $\zeta_0 \in \mathbb{R}$, $(\zeta_1, \ldots, \zeta_{n-1}) \in \mathbb{R}_+^{n-1}$ and $\sum_{j=0}^{n-1} \zeta_j \leq \max_i y_i$.

Let $\nabla g(\zeta)$ denote the gradient of function $g$ at point $\zeta$.

It should be noted that for larger-scale problems the solution can appear computationally quite challenging. In this regard, the present study proposes to use a greedy algorithm of Frank–Wolfe type for solving this problem.

The following (Algorithm 2) is a pseudocode of Frank–Wolfe-type algorithm for the problem (2).

The rate of convergence is estimated according to the following theorem.
Algorithm 2: Greedy algorithm for sparse monotone regression

begin
  · Let \( N \) be the number of iteration;
  · Our function \( g(\zeta) \) and the feasible set \( S \) were defined above.
  
  Let \( \nabla g(\zeta) = \left( \frac{\partial g}{\partial \zeta_0}, \frac{\partial g}{\partial \zeta_1}, \ldots, \frac{\partial g}{\partial \zeta_{n-1}} \right) \) be the gradient of function \( g \) at a point \( \zeta \),
    \[ \frac{\partial g}{\partial \zeta_k} = \frac{2}{n} \sum_{i=k}^{n} \left( \sum_{j=0}^{i-1} \zeta_j - y_i \right), \quad k = 0, \ldots, n-1; \]
  · Let zero vector \( \zeta^0 = (0, \ldots, 0) \) be the start point, and let the counter \( t = 0 \);
  · while \( t < N \) do
    · Calculate \( \nabla g(\zeta^t) \), the gradient of the function \( g \) at the point \( \zeta^t \);
    · Let \( \tilde{\zeta}^t \) be the solution of the linear optimization problem
      \( \langle \nabla g(\zeta^t)^T, \zeta \rangle \rightarrow \min \quad \zeta \in S \),
      where \( \langle \nabla g(\zeta^t)^T, \zeta \rangle \) is a scalar product of vectors;
    · (Update step) Let \( \zeta^{t+1} = \zeta^t + \alpha_t (\tilde{\zeta}^t - \zeta^t) \), \( \alpha_t = \frac{2}{t+2} \) and than \( t := t + 1 \);
    · Recover the monotone sequence \( z = (z_1, \ldots, z_n) \) from the vector of increments \( \zeta^N \);
end

Theorem 1. Let \( \{\zeta^t\} \) be generated according to the Frank–Wolfe method (Algorithm 2) using the step-size rule \( \alpha_t = \frac{2}{t+2} \). Then for all \( t \geq 2 \)

\[
g(\zeta^t) - g^* \leq 4\sqrt{\frac{n(n+1)(2n+1)}{6n^2}} \left( \max_i y_i - \min_i y_i \right)^2 \frac{1}{t+2},
\]

where \( g^* \) is the optimal solution of (2).

Proof. It is know [18] that for all \( t \geq 2 \):

\[
g(\zeta^t) - g^* \leq \frac{2L(Diam(S))^2}{t+2},
\]

where \( L \) is the Lipschitz constant and \( Diam(S) \) is the diameter of \( S \).

Let

\[
\nabla^2 g(\zeta) := \left( \frac{\partial^2 g}{\partial \zeta_0^2}, \frac{\partial^2 g}{\partial \zeta_1^2}, \ldots, \frac{\partial^2 g}{\partial \zeta_{n-1}^2} \right).
\]

It is well-known that if \( \nabla g \) is differentiable then its Lipschitz constant \( L \) satisfies the inequality

\[
L \leq \sup_\zeta \| \nabla^2 g(\zeta) \|_2.
\]
Then

\[
L \leq \sup_\zeta \sqrt{\sum_{k=0}^{n-1} \left( \frac{\partial^2 g}{\partial \zeta^2} \right)^2} = \\
= \frac{1}{n} \left( \sum_{k=1}^{n} (2(n - k + 1))^2 \right) = \frac{2}{n} \sum_{k=1}^{n} k^2 = 2\sqrt{\frac{n(n+1)(2n+1)}{6n^2}},
\]

(4)

It is easy to prove and \( \text{Diam}(S) := \sqrt{2}(\max_i y_i - \min_i y_i) \).

The disadvantage of this method is the dependence of the theoretical degree of convergence on the dimensionality of the problem. The papers [28], [20], [29] suggest to use the values of duality gap as the stopping criterion for Frank-Wolfe type algorithms.

### 3 Empirical Result

The algorithms have been implemented both in R and C++. We compared the performance of the greedy algorithm (Algorithm 2) with the performance of PAVA (Algorithm 1) using simulated data sets.

It should also be noted that the PAVA’s speed is significantly higher for small-scale tasks in R. But if the number of points is greater than at least 2000, the greedy algorithm spends less time searching for a solution (Fig. 1).

Tables 1, 2 present empirical results for PAVA and greedy algorithms for a simulated set of points. The simulated points are obtained as the values of logarithm function with added normally distributed noise: \( A = \{(x_i, y_i), y_i = \ln(x_0 + i\Delta x) + \varphi_i, \varphi_i \sim N(0, 1), x_0 = 1, \Delta x = 1, i = 1, \ldots, 10000 \} \). The dimension of the problem is 10000 points. The tables contain information on errors \( \frac{1}{n} \sum_{i=1}^{n} (z_i - y_i)^2 \), elapsed time, cardinality and greedy algorithm’s iteration number.

The results show that error of greedy algorithm are getting closer to the error of PAVA with increase of number of iterations for greedy algorithm. While PAVA is better than greedy algorithm in terms of errors, the solutions of greedy algorithm have a better sparsity. Greedy algorithm’s output solution is more sparse. It should be noted that the elapsed time for PAVA implemented in C++ is smaller than for greedy algorithm. However, greedy algorithm has a better rate of convergence if number of iterations is less than 700 for the algorithms implemented in R., Both algorithms obtain a sparse solutions, but we can control the number of nonzero elements (cardinality) in the greedy algorithm as opposed to PAVA. Generally, the greedy algorithm’s cardinality increases by one at each iteration. Consequently, we should limit the number of iterations to obtain more sparse solution.

Figure 2 shows simulated points \( (N = 100) \) with logarithm structure and isotonic regressions, where green line represents the greedy algorithm’s isotonic
Fig. 1. The dependence of the CPU time on dimension of the problem of the greedy algorithm (green line, 100 iterations) and PAVA (red line) implemented in R.

Table 1. Comparison of algorithms PAVA and greedy algorithm (Greedy) on an example of the simulated data (implementation in language C++):

\[ A = \{(x_i, y_i), y_i = \ln(x_0 + i\Delta x) + \varphi_i, \varphi_i \sim N(0, 1), x_0 = 1, \Delta x = 1, i = 1, \ldots, 10000\]  

<table>
<thead>
<tr>
<th>Algorithm (the number of iterations)</th>
<th>Error</th>
<th>Cardinality</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAVA</td>
<td>0.994</td>
<td>82</td>
<td>0.00</td>
</tr>
<tr>
<td>Greedy (10)</td>
<td>1.254</td>
<td>7</td>
<td>0.00</td>
</tr>
<tr>
<td>Greedy (50)</td>
<td>1.101</td>
<td>31</td>
<td>0.00</td>
</tr>
<tr>
<td>Greedy (100)</td>
<td>1.003</td>
<td>43</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy (200)</td>
<td>1.001</td>
<td>55</td>
<td>0.01</td>
</tr>
<tr>
<td>Greedy (500)</td>
<td>0.995</td>
<td>74</td>
<td>0.03</td>
</tr>
<tr>
<td>Greedy (1000)</td>
<td>0.995</td>
<td>78</td>
<td>0.07</td>
</tr>
<tr>
<td>Greedy (2000)</td>
<td>0.994</td>
<td>82</td>
<td>0.09</td>
</tr>
<tr>
<td>Greedy (5000)</td>
<td>0.994</td>
<td>82</td>
<td>0.17</td>
</tr>
<tr>
<td>Greedy (10000)</td>
<td>0.994</td>
<td>82</td>
<td>0.33</td>
</tr>
</tbody>
</table>

regression and red line presents PAVA’s isotonic regression. Greedy algorithm gives a solution with 14 jumps, and PAVA with 16 jumps. Since the solutions of the greedy algorithm are more sparse, the greedy algorithm error \( \varepsilon \) is slightly higher than the PAVA.

The obtained empirical results for the greedy algorithm show that the degree of convergence for the considered examples is much higher than its theoretical estimates obtained in Theorem 1.
Table 2. Comparison of algorithms PAVA and greedy algorithm (Greedy) on an example of the simulated data (implementation in language R): $A = (x_i, y_i)$

<table>
<thead>
<tr>
<th>Algorithm (the number of iterations)</th>
<th>Error</th>
<th>Cardinality</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAVA</td>
<td>0.994</td>
<td>82</td>
<td>4.28</td>
</tr>
<tr>
<td>Greedy (10)</td>
<td>1.169</td>
<td>6</td>
<td>0.09</td>
</tr>
<tr>
<td>Greedy (50)</td>
<td>1.011</td>
<td>30</td>
<td>0.33</td>
</tr>
<tr>
<td>Greedy (100)</td>
<td>0.999</td>
<td>41</td>
<td>0.63</td>
</tr>
<tr>
<td>Greedy (200)</td>
<td>0.996</td>
<td>57</td>
<td>1.23</td>
</tr>
<tr>
<td>Greedy (500)</td>
<td>0.995</td>
<td>76</td>
<td>3.08</td>
</tr>
<tr>
<td>Greedy (1000)</td>
<td>0.994</td>
<td>79</td>
<td>6.28</td>
</tr>
<tr>
<td>Greedy (2000)</td>
<td>0.994</td>
<td>82</td>
<td>12.67</td>
</tr>
<tr>
<td>Greedy (5000)</td>
<td>0.994</td>
<td>82</td>
<td>31.58</td>
</tr>
<tr>
<td>Greedy (10000)</td>
<td>0.994</td>
<td>82</td>
<td>60.91</td>
</tr>
</tbody>
</table>

Fig. 2. Step functions obtained by the greedy algorithm ($\varepsilon = 0.753$) and PAVA ($\varepsilon = 0.751$)

4 Conclusion

Our research proposes an algorithm for solving the problem of constructing the best fitted monotone regression by using the Frank–Wolfe method. The software was implemented in R and C++. We compared the performance of the greedy algorithm with the performance of PAVA using simulated data sets. While PAVA gives a slightly smaller errors than greedy algorithm, greedy algorithm obtains significantly sparser solutions. The advantages of greedy algorithm are the simplicity of implementation, the potential for controlling cardinality and the
elapsed time is lower for the implementation in R in the case of problem with large dimension.

References

2. Bach, F.: Efficient algorithms for non-convex isotonic regression through submodular optimization. Tech. Rep. hal-01569934 (Jul 2017), https://hal.archives-ouvertes.fr/hal-01569934, working paper or preprint
27. Sidorov, S.P.: Linear k-monotonicity preserving algorithms and their approximation properties. LNCS 9582, 93–106 (2016)