Appendix to "A path algorithm for the Fused Lasso Signal Approximator" published in the Journal of Computational and Graphical Statistics

Holger Hoefling
Institute of Medical Biometry and Medical Informatics
Dept. of Medical Biometry and Statistics
University Medical Center Freiburg

6 Introduction

In this online supplement to the article "A path algorithm for the Fused Lasso Signal Approximator", we will give proofs and further results that would have gone beyond the possible size and scope of the paper. This Supplement will be structured as follows.

In Section 7, we give a brief overview of subgradients and the relevant calculations that we need for our algorithms. After this, the proofs to Theorem 2 and Proposition 1 as well as the complexity calculations and some tricks for the maximum-flow computations are given in Section 8. The proofs and complexity calculations for the special one-dimensional FLSA will be treated in Section 9.

7 Introduction to subgradients

In convex optimization, often some of the functions in a problem are not differentiable everywhere. In this case, instead of a regular gradient, we can use a subgradient. The following introduction is mostly taken from Bertsekas [1999]. As a starting point, we first want to define what a subgradient is and then describe a condition that guarantees that a convex function has a minimum at a point x. Afterwards, we will derive the relevant subgradient expressions that are being used in this article.

Definition 2. *Given a convex function $f : \mathbb{R}^n \to \mathbb{R}$, we say a vector $d \in \mathbb{R}^n$ is a* subgradient *of f at a point $x \in \mathbb{R}^n$ if*

$$f(z) \geq f(x) + (z - x)'d, \quad \forall\, z \in \mathbb{R}^n.$$

If instead f is a concave function, we say that d is a subgradient of f if $-d$ is a subgradient of $-f$ at x. The set of all subgradients of a convex function f at $x \in \mathbb{R}^n$ is called the subdifferential of f at x, and is denoted by $\partial f(x)$.

For subgradients, some basic properties similar to regular gradients hold and the proof to the following Proposition can be found in Bertsekas [1999] on pp. 712-716.

Proposition 2. *Let $f : \mathbb{R}^n \to \mathbb{R}$ be convex. For every $x \in \mathbb{R}^n$, the following hold:*

1. *If f is equal to the sum $f_1 + \cdots + f_m$ of convex functions $f_j : \mathbb{R}^n \to \mathbb{R}, j = 1, \ldots, m$, then $\partial f(x)$ is equal to the vector sum $\partial f_1(x) + \cdots + \partial f_m(x)$.*

2. *If f is equal to the composition of a convex function $h : \mathbb{R}^m \to \mathbb{R}$ and an $m \times n$ matrix $\mathbf{A}$, that is $[f(x) = h(\mathbf{A}x)]$, then $\partial f(x)$ is equal to $\mathbf{A}'\partial h(\mathbf{A}x) = \{\mathbf{A}'g : g \in \partial h(\mathbf{A}x)\}$.*

3. *x minimizes f over a convex set $\mathcal{A} \subset \mathbb{R}^n$ if and only if there exists a subgradient $d \in \partial f(x)$ such that*
$$d'(z - x) \geq 0. \quad \forall z \in \mathcal{A}.$$

In our case here for the convex functions we optimize over the set $\mathcal{A} = \mathbb{R}^n$ and therefore the last statement says that x minimizes a function f over $\mathbb{R}^n$ if and only if a subgradient $d \in \partial f(x)$ exists such that
$$d = 0.$$
Now, by using the proposition from above, all we need to calculate the subgradient of the loss function is the subgradient of $f(x) = |x|$ which is $\partial f(0) = [-1, 1]$ and $\partial f(x) = \text{sign}(x)$ for $x \neq 0$. Therefore, the subgradient of the loss function
$$\frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^T(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \lambda_1 \sum_{i=1}^{p} |\beta_i| + \lambda_2 \sum_{(i,j)\in E, i<j} |\beta_i - \beta_j|$$
w.r.t. β_i is
$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})_i + \lambda_1 \sum_{i=1}^{p} \frac{\partial f(\beta_i)}{\partial \beta_i} + \lambda_2 \sum_{(i,j)\in E, i<j} \frac{\partial f(\beta_i - \beta_j)}{\partial \beta_i}$$
and using the optimality condition from above, a solution $\boldsymbol{\beta}$ is optimal if there exists s_i, t_{ij} such that
$$(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})_i + \lambda_1 s_i + \lambda_2 \sum_{(i,j)\in E, i<j} t_{ij} \quad \text{for} \quad i = 1, \ldots, p$$
where $s_i = \text{sign}(\beta_i)$ for $\beta_i \neq 0$ and $s_i \in [-1, 1]$ otherwise. Similarly $t_{ij} = \text{sign}(\beta_i - \beta_j)$ for $\beta_i = \beta_j$ and $t_{ij} \in [-1, 1]$ otherwise. For notational convenience, we set $t_{ji} = -t_{ij}$ for all $i < j$. Setting $\mathbf{X} = \mathbf{I}$ and $\lambda_1 = 0$ gives the subgradients for the FLSA.

8 The Fused Lasso Signal Approximator with general graph

In this section we want to give the proofs of Theorem 2 and Proposition 1 of the main article. First, we will turn our attention to the Theorem and then to the Proposition. For the following proofs please note that only the existence of a solution with any $\hat{t}_{kl}(\lambda_2)$ is required and none of the proofs assumes that they have to be unique. Therefore, despite a possible identifiability issue w.r.t. $\hat{t}_{kl}(\lambda_2)$, the proofs are valid and guarantee that the path algorithm works.

8.1 Proof of Theorem 2

Proof. In order to show that this is indeed the solution, we have to show that for any $\lambda_2 \in [\lambda_2^0, \lambda_2^0 + \Delta]$, the subgradient Equations (5) hold. We will do this for every group $F_i(\lambda_2)$ separately. We know that

$$\frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2) = -\frac{1}{|F_i(\lambda_2)|} \sum_{j \neq i} \sum_{(k,l) \in E; k \in F_i(\lambda_2); l \in F_j(\lambda_2)} \hat{t}_{kl}(\lambda_2).$$

For the maximum flow graph, we know that for all nodes except the source and the sink, the same amount flows into and out of each node, resulting in a net flow of 0. Furthermore, we also know that the sum of all capacities coming from the source is the same as the sum of all capacities going to the sink as

$$\sum_{(r,l) \in \tilde{E}_i(\lambda_2)} c_{rl} - \sum_{(k,s) \in \tilde{E}_i(\lambda_2)} c_{ks} = \sum_{k \in F_i(\lambda_2)} \hat{p}_k(\lambda_2) = 0$$

which we get immediately when plugging in the definition of $\hat{p}_k(\lambda_2)$ and the fact that for any node that is not connected to either the source or the sink, $\hat{p}_k(\lambda_2) = 0$. From this we can see that if all flows coming from the source are at maximum capacity, so are all flows going to the sink. Therefore, for a maximum flow with all flows from the source being at maximum capacity, we have for all $k \in F_i(\lambda_2)$

$$\sum_{l:(k,l) \in E; l \in F_i(\lambda_2)} \hat{f}_{kl}(\lambda_2) = \begin{cases} -\hat{f}_{kr}(\lambda_2) = c_{rk}(\lambda_2) = \hat{p}_k(\lambda_2) & \text{if } (r,k) \in \tilde{E}_i(\lambda_2) \\ -\hat{f}_{ks}(\lambda_2) = -c_{ks}(\lambda_2) = \hat{p}_k(\lambda_2) & \text{if } (k,s) \in \tilde{E}_i(\lambda_2) \\ 0 = \hat{p}_k(\lambda_2) & \text{otherwise} \end{cases}$$

so that overall we get

$$\hat{\beta}_k(\lambda_2) - y_k + \lambda_2 \sum_{l:(k,l) \in E} \hat{t}_{kl}(\lambda_2)$$

$$= \hat{\beta}_k(\lambda_2^0) - y_k + \lambda_2 \sum_{l:(k,l)\in E} \hat{t}_{kl}(\lambda_2^0) + \frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2) \cdot (\lambda_2 - \lambda_2^0)$$

$$+ \sum_{l\in F_i(\lambda_2):(k,l)\in E} \hat{f}_{kl}(\lambda_2) \cdot (\lambda_2 - \lambda_2^0) + \sum_{l\notin F_i(\lambda_2):(k,l)\in E} \hat{t}_{kl}(\lambda_2^0) \cdot (\lambda_2 - \lambda_2^0)$$

$$= (\lambda_2 - \lambda_2^0) \left[\frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2) + \sum_{l\in F_i(\lambda_2):(k,l)\in E} \hat{f}_{kl}(\lambda_2) + \sum_{l\notin F_i(\lambda_2):(k,l)\in E} \hat{t}_{kl}(\lambda_2^0) \right]$$

$$= (\lambda_2 - \lambda_2^0) \left[\sum_{l\in F_i(\lambda_2):(k,l)\in E} \hat{f}_{kl}(\lambda_2) - \hat{p}_k(\lambda_2) \right] = 0$$

where we use

$$\hat{\beta}_k(\lambda_2^0) - y_k + \lambda_2^0 \sum_{l:(k,l)\in E} \hat{t}_{kl}(\lambda_2^0) = 0$$

by assumption in the second equality and the definition of $\hat{p}_k(\lambda_2^0) = -\sum_{l\notin F_i(\lambda_2^0):(k,l)\in E} \hat{t}_{kl}(\lambda_2^0) - \frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2^0)$ in the third. As this is true for every $k \in F_i(\lambda_2)$ and group $F_i(\lambda_2)$, the solution as proposed in the theorem solves the subgradient equations and therefore minimizes the loss function. This holds for all $\lambda_2 \in [\lambda_2^0, \lambda_2^0 + \Delta]$ by the construction of Δ as in the interval $[\lambda_2^0, \lambda_2^0 + \Delta]$, the sets $F_i(\lambda_2$ stay the same and therefore all of the equations considered above stay the same as well. □

8.2 Proof of Proposition 1

Proof. First, we want to show that after the fusion and splitting steps finish, we have a valid set of fused variables according to Definition 1. For this, we have to address two issues:

First, we have to show that for sets $F_i(\lambda_2)$ and $F_j(\lambda_2)$ it holds that $\hat{\beta}_{F_i}(\lambda_2) \neq \hat{\beta}_{F_j}(\lambda_2)$ for $\lambda_2 \in (\lambda_2^0, \lambda_2^0 + \varepsilon)$ for some $\varepsilon > 0$. However, this is certainly true. If $\hat{\beta}_{F_i}(\lambda_2^0) \neq \hat{\beta}_{F_j}(\lambda_2^0)$, then this holds also for some interval $\lambda_2 \in (\lambda_2^0, \lambda_2^0 + \varepsilon)$ as $\hat{\beta}_k(\lambda_2)$ is continuous in λ_2. If we have that $\hat{\beta}_{F_i}(\lambda_2^0) = \hat{\beta}_{F_j}(\lambda_2^0)$, then we can assume that $\frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2^0) > \frac{\partial \hat{\beta}_{F_j}}{\partial \lambda_2}(\lambda_2^0)$ and $\hat{t}_{kl}(\lambda_2^0) = 1$ for $k \in F_i(\lambda_2^0), l \in F_j(\lambda_2^0); (k,l) \in E$ (otherwise $F_i(\lambda_2^0)$ and $F_j(\lambda_2^0)$ would have been fused by rule 2). However, this means that $\hat{\beta}_{F_i}(\lambda_2) > \hat{\beta}_{F_j}(\lambda_2)$ for $\lambda_2 \in (\lambda_2^0, \lambda_2^0 + \varepsilon)$ due to the derivative and that $\hat{t}_{kl}(\lambda_2) = \text{sign}(\hat{\beta}_{F_i}(\lambda_2) - \hat{\beta}_{F_j}(\lambda_2))$, so the grouping is valid.

Second, for all of the sets at the end of the splitting steps, the maximal flow condition of Theorem 2 trivially holds as any set for which it doesn't hold would be split up into two smaller sets (and it always holds for sets of size 1).

Therefore, it only remains to show that the fusion and splitting steps converge after a finite number of iterations. For this, we show that if a set $F_i(\lambda_2^0)$ was split into subgroups $R_i(\lambda_2^0)$ and $S_i(\lambda_2^0)$ at penalty parameter λ_2^0, then $R_i(\lambda_2^0)$ and $S_i(\lambda_2^0)$ will not be merged at λ_2^0 in a subsequent iteration. From this, we will conclude that there is only a finite number

of possible iterations, as there is only a finite number of possible sets and we cannot have infinite cycles of fusions and splits for the sets. Therefore, the algorithm converges after a finite number of steps.

So, as $F_i(\lambda_2^0-)$ was split into $R_i(\lambda_2^0)$ and $S_i(\lambda_2^0)$, we know that $R_i(\lambda_2^0) \neq \emptyset$ as well as $S_i(\lambda_2^0) \neq \emptyset$. For set $R_i(\lambda_2^0)$ consider the capacities of edges from the source into $R_i(\lambda_2^0)$ minus the capacities of edges going from $R_i(\lambda_2^0)$ either to $S_i(\lambda_2^0)$ or directly to the sink. As we split group $F_i(\lambda_2^0-)$, we know that the capacities going into $R_i(\lambda_2^0)$ are larger than the capacities going out as otherwise the source-edges would be at maximum capacity and therefore,

$$\sum_{(r,l)\in\tilde{E}_i(\lambda_2^0);l\in R_i(\lambda_2^0)} c_{rl}(\lambda_2^0) > \sum_{(k,l)\in\tilde{E}_i(\lambda_2^0);k\in R_i(\lambda_2^0),l\in S_i(\lambda_2^0)} c_{kl}(\lambda_2^0) + \sum_{(k,s)\in\tilde{E}_i(\lambda_2^0);k\in R_i(\lambda_2^0)} c_{ks}(\lambda_2^0).$$

For all edges $(k,l) \in \tilde{E}_i(\lambda_2^0)$ with $k \in R_i(\lambda_2^0)$ and $l \in S_i(\lambda_2^0)$ we also know that $1 = c_{kl}(\lambda_2^0) = \hat{f}_{kl}(\lambda_2^0) = \hat{t}_{kl}(\lambda_2^0)$ (because the only other possibility is $c_{kl}(\lambda_2^0) = \infty$ and this is not possible by definition of $R_i(\lambda_2^0)$ and $S_i(\lambda_2^0)$). Here $\hat{f}_{kl}(\lambda_2^0) = c_{kl}(\lambda_2^0)$ follows as otherwise l would be connected to the source in the residual graph and therefore $l \in R_i$, which is not the case as $l \in S_i$. Furthermore $\hat{t}_{kl}(\lambda_2^0) = 1$ as $c_{kl}(\lambda_2^0) = 1$ (by definition of the capacities). Then

$$\sum_{(r,l)\in\tilde{E}_i(\lambda_2^0);l\in R_i(\lambda_2^0)} c_{rl}(\lambda_2^0) - \sum_{(k,s)\in\tilde{E}_i(\lambda_2^0);k\in R_i(\lambda_2^0)} c_{ks}(\lambda_2^0) - \sum_{(k,l)\in\tilde{E}_i(\lambda_2^0);k\in R_i(\lambda_2^0),l\in S_i(\lambda_2^0)} \hat{t}_{kl}(\lambda_2^0) > 0.$$

Using that $c_{rl}(\lambda_2^0) = \hat{p}_l(\lambda_2^0)$ for $(r,l) \in \tilde{E}_i$ and $c_{ks}(\lambda_2^0) = -\hat{p}_k(\lambda_2^0)$ for $(k,s) \in \tilde{E}_i(\lambda_2^0)$, we get

$$\sum_{k\in R_i(\lambda_2^0)} \hat{p}_k(\lambda_2^0) - \sum_{(k,l)\in\tilde{E}_i(\lambda_2^0);k\in R_i(\lambda_2^0),l\in S_i(\lambda_2^0)} \hat{t}_{kl}(\lambda_2^0) > 0. \tag{12}$$

Let $p_k^{R_i}(\lambda_2^0)$ be the push on node k in the graph associated with group $R_i(\lambda_2^0)$ and $p_l^{S_i(\lambda_2^0)}$ the push on node l associated with $S_i(\lambda_2^0)$ after the split. From Equation (9), we know that

$$\sum_{k\in R_i(\lambda_2^0)} \hat{p}_k^{R_i}(\lambda_2^0) = 0.$$

We can also infer from the definition of the push on $F_i(\lambda_2^0)$ and $R_i(\lambda_2^0)$ that

$$\sum_{k\in R_i(\lambda_2^0)} \hat{p}_k^{R_i}(\lambda_2^0) + |R_i(\lambda_2^0)|\frac{\partial\hat{\beta}_{R_i}}{\partial\lambda_2}(\lambda_2^0) = \sum_{k\in R_i(\lambda_2^0)} \hat{p}_k(\lambda_2^0) - \sum_{k\in R_i(\lambda_2^0),l\in S_{i,}(\lambda_2^0),(k,l)\in E} \hat{t}_{kl}(\lambda_2^0)$$
$$+ |R_i(\lambda_2^0)|\frac{\partial\hat{\beta}_{F_i}}{\partial\lambda_2}(\lambda_2^0)$$

and therefore using Equation (12) and $\sum_{k \in R_i(\lambda_2^0)} p_k^{R_i}(\lambda_2^0) = 0$ that

$$\frac{\partial \hat{\beta}_{R_i}}{\partial \lambda_2}(\lambda_2^0) > \frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2^0).$$

Now, again using Equation (9), we get

$$\frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2^0) = \frac{|R_i(\lambda_2^0)|}{|R_i(\lambda_2^0)| + |S_i(\lambda_2^0)|} \frac{\partial \hat{\beta}_{R_i}}{\partial \lambda_2}(\lambda_2^0) + \frac{|S_i(\lambda_2^0)|}{|R_i(\lambda_2^0)| + |S_i(\lambda_2^0)|} \frac{\partial \hat{\beta}_{S_i}}{\partial \lambda_2}(\lambda_2^0)$$

and thus

$$\frac{\partial \hat{\beta}_{R_i}}{\partial \lambda_2}(\lambda_2^0) > \frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2^0) > \frac{\partial \hat{\beta}_{S_i}}{\partial \lambda_2}(\lambda_2^0).$$

As also $\hat{t}_{kl}(\lambda_2^0) = 1$ for $k \in R_i(\lambda_2^0)$ and $l \in S_i(\lambda_2^0)$, we can see that the groups that have just been split at λ_2^0 cannot be merged again at the same penalty parameter λ_2^0 as the merging condition is violated due to the previous result $\partial \hat{\beta}_{R_i}(\lambda_2)/\partial \lambda_2 > \partial \hat{\beta}_{S_i}/\partial \lambda_2$.

In turn, this also means that any two sets that have just been fused, cannot be immediately split up again into the same original sets at λ_2^0 as this would lead us back to the starting position, which would force us to fuse the two sets again. However, we have just shown that this cannot happen and therefore we also see that any two sets that are being fused, are not being split up again immediately. This shows that after the splitting and fusion steps, the resulting sets are valid. Therefore, Theorem 2 can be applied again, guaranteeing an affine solution for an interval of positive length. □

8.3 Complexity

The computational complexity of the algorithm is dominated by the complexity of the calculations for the max-flow problem, which when using the Dinitz blocking flow algorithm with dynamic trees (see Sleator and Tarjan [1983]) has a complexity of $O(|V||E| \log(|V|)$ where $|V|$ is the number of nodes in the graph and $|E|$ is the number of edges in the graph. For simplicity, we assume that we are dealing with graphs for which the degree of each node is bounded and independent of the number of nodes $|V|$ in the graph (i.e. for the 2-dimensional grid, each node has degree of 4 or less). Under this assumption, the complexity of the max-flow algorithm is then $O(|V|^2 \log(|V|)$. Because in general it is hard to calculate the complexity of the algorithm, we will next look at a *good* case and a *bad* case scenario.

8.3.1 Good case scenario

As the size of the graphs for which we calculate the maximum flows is important, and knowing that for large λ_2, all variables will be fused, the optimal scenario occurs when the average size of the fused sets in the algorithm remains small. This is the case when no splits occur

and the two smallest nodes always merge. Then, when we describe the merges in the form of a tree, we get a balanced binary tree of depth $\lceil \log_2(n) \rceil$. In this case, the total complexity of the algorithm is

$$\begin{aligned}
\text{Complexity} &= C_1 \sum_{\text{k=depth of tree}} \#\text{ of nodes at depth k} \cdot \\
&\quad (\text{set size at depth k})^2 \cdot \log(\text{set size at depth k}) \\
&\leq C_2 \sum_{k=1}^{\lceil \log_2(n) \rceil} 2^{\lceil \log_2(n) \rceil - k + 1} \cdot 2^{2k-2} \cdot k \\
&\leq C_3 n \sum_{k=1}^{\lceil \log_2(n) \rceil} k 2^k \leq C n^2 \log(n)
\end{aligned}$$

where C_k are all positive constants. Therefore, the complexity in this case is on the order $O(n^2 \log(n))$.

8.3.2 Bad case scenario

In this scenario we assume that at each merge is between the already largest set and another set of size one. So, for the $k-th$ merge a set of size k and one of size 1 will be merged. This way, the average size of the sets is $n/2$ throughout the algorithm - substantially larger than in the good case scenario. This order of merges is the worst possible constellation. With respect to possible splits of sets, we assume that for roughly 2 merges, one split occurs (i.e. a set is merged, then splits and is then merged again). Under this assumption, the number of splits only grows linearly with n. Theoretically, it is possible that with larger n, over-proportionally more splits could occur. Then the complexity of the algorithm would be worse than what we have in this scenario. Overall we have then

$$\begin{aligned}
\text{Complexity} &= C_1 \sum_{k=1}^{n} \lceil \log_2(k) \rceil k^2 \\
&\leq C n^3 \log(n)
\end{aligned}$$

so that the complexity is $O(n^3 \log(n))$.

8.3.3 Approximate algorithm

In the approximate algorithm, whenever a set is of size K or larger, we do not execute the max-flow algorithm anymore. All other computations (which we neglected before as the max-flow algorithm dominates them) associated with the fused set are of complexity $O(\log(n))$

(incremental cost of finding $\min h_{ij}$ and $\min v_{ij}$). Therefore, in the good case we now have

$$\begin{aligned}\text{Complexity} =& C_1(\#\text{ of merges} \cdot \log(n) + \sum_{\text{k=depth of tree}} \text{number of nodes at depth k} \cdot \\ & (\text{set size at depth k if} < \text{K})^2 \cdot \log(\text{set size at depth k if} < \text{K})) \\ &= C_2 \left(n \log(n) + \sum_{k=1}^{\lceil \log_2(K) \rceil} 2^{\lceil \log_2(n) \rceil - k + 1} \cdot k 2^{2k-2} \right) \\ &\leq C_3 \left(n \log(n) + nK \log(K) \right) \leq CnK \log(n)\end{aligned}$$

and thus a complexity of $O(Kn \log(n))$ whereas in the bad case

$$\begin{aligned}\text{Complexity} &= C_1 \left(n \log(n) + \sum_{k=1}^{K} \lceil \log_2(k) \rceil k^2 \right) \\ &\leq CK^2 n \log(n)\end{aligned}$$

we get $O(K^2 n \log(n))$.

8.4 Solving the linear program using iterative maximum-flows

In Section 2.3.4, we presented the linear program in Equation (10) which we have to solve in order to determine the exact value of λ_2 at which the set has to be split. As we already mentioned there, we can either use a standard convex programming procedure to solve it or an iterative procedure based on solving maximum-flow problems. Here, we will present the procedure based on the maximum flow problem.

Assume that we are currently at penalty parameter λ_2^0 for set $F_i(\lambda_2^0)$ with associated maximum flow graph $\tilde{\mathcal{G}}(\lambda_2^0, \delta)$. For this graph define the sum of all source capacities as

$$sc(\lambda_2^0) = \sum_{(r,l) \in \tilde{E}_i(\lambda_2^0)} c_{rl}$$

which is independent of δ and the sum of source residuals as

$$sr(\lambda_2^0, 1/\delta) = \sum_{(r,l) \in \tilde{E}_i(\lambda_2^0)} \hat{f}_{rl}(\lambda_2^0, \delta) - sc(\lambda_2^0)$$

which depends on δ as the maximum flow in the graph f depends on δ.

For the following first note that the maximum flow in the graph $\tilde{\mathcal{G}}(\lambda_2^0, \delta)$ is a piecewise linear, convex function in $1/\delta$. Here, the piecewise linearity follows from the linearity of the capacities in $1/\delta$. The convexity is easy to show using the fact that the constraints are affine.

Therefore, $sr(\lambda_2^0, 1/\delta)$ is a piecewise linear, decreasing, convex function. Our goal is to find the minimal $1/\delta$ for which $sr(\lambda_2^0, 1/\delta)$ is 0.

For the start of the algorithm define $\delta^1 := h_i(\lambda_2^0) - \lambda_2^0 + \varepsilon$, for which we know that $sr(\lambda_2^0, \delta^1) > 0$ (otherwise, we would not have needed to start the iterative algorithm). Furthermore, set $\delta^0 = \infty$. Then we use the following very simple root searching scheme:

Algorithm 3: Iterative linear programming solution using maximum-flow problems

initialize
  Set $k := 1$;
end
while $sr(\lambda_2^0, 1/\delta^k) > 0$ **do**
  $d^k := \frac{sr(\lambda_2^0, 1/\delta^k) - sr(\lambda_2^0, 1/\delta^{k-1})}{1/\delta^k - 1/\delta^{k-1}}$;
  $\frac{1}{\delta^{k+1}} := \frac{1}{\delta^k} - \frac{sr(\lambda_2^0, 1/\delta^k)}{d^k}$;
  $k := k + 1$;
end

Lemma 1. *The Algorithm 3 converges in a finite number of steps to the root of function* $sr(\lambda_2^0, 1/\delta)$.

Proof. To see this, we use the piecewise linearity of the function. Assume that we are at iteration k. Let $1/\delta_k \in [a_k, b_k]$ such that $sr(\lambda_2^0, 1/\delta)$ is linear on $[a_k, b_k]$ and a_k and b_k are the breakpoints of the linear function.

Case 1: If $sr(\lambda_2^0, b_k) = 0$ is the root we are searching for and $1/\delta^{k-1} \in [a_k, b_k]$, then by construction we have that $1/\delta^{k+1} = b_k$ and we are finished.

Case 2: Assume $1/\delta^{k-1} \in [a_{k-1}, b_{k-1}] \neq [a_k, b_k] \ni 1/\delta^k$. Then, as $sr(\lambda_2^0, 1/\delta^k) > 0$ and $d_k < 0$ (decreasing convex function), it follows that $1/\delta^{k+1} > 1/\delta^k \notin [a_{k-1}, b_{k-1}]$. That is, $1/\delta^{k+1}$ is always at least one interval closer to the root than $1/\delta^{k-1}$.

Case 3: Assume we have $1/\delta^{k-1}, 1/\delta^k \in [a_k, b_k]$ where b_k is not the root. Then d_k is the derivative of $sr(\lambda_2^0, 1/\delta)$ on $[a_k, b_k]$ and as $sr(\lambda_2^0, b_k) > 0$, we have $1/\delta^{k+1} > b_k$, i.e. again $1/\delta^{k+1}$ is in an interval that is closer to the root than $1/\delta^{k-1}$.

With these 3 cases, we have shown that the sequence $1/\delta^k$ is always moving closer to the root (there are only finitely many intervals where $sr(\lambda_2^0, 1/\delta)$ is piecewise linear) and therefore in a finite number of steps, we have that $1/\delta^{k-1}, 1/\delta^k \in [a_k, b_k]$ where b_k is the root. But then by case 1, $b_k = 1/\delta^{k+1}$ and we have converged to the solution in a finite number of steps. □

In order to estimate the complexity that is added to the algorithm by this iterative approach, we have counted the number of iterations that were used when applying the algorithm to the $200x200$ two-dimensional problem described in Section 2.6. A plot depending of the

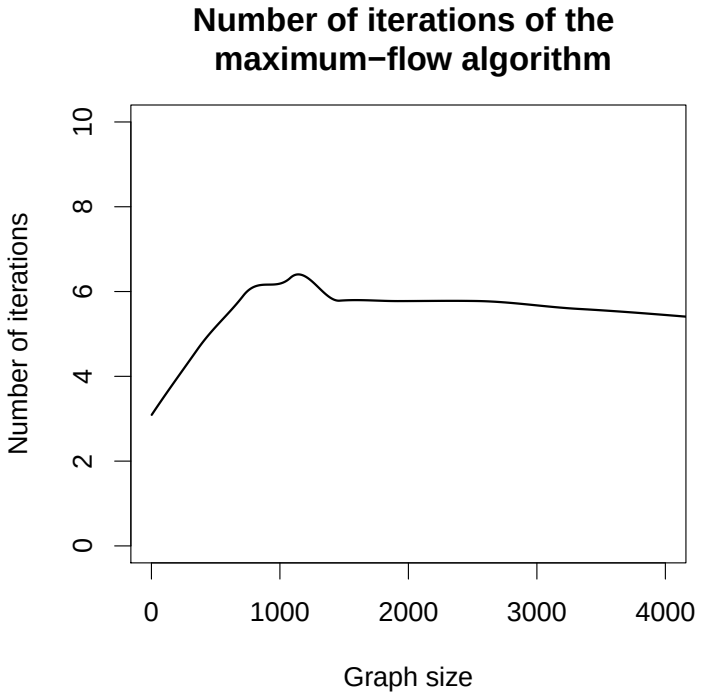

Figure 4: Number of iterations needed until convergence for the iterative maximum-flow approach.

size of the graph to which the iterative maximum-flow approach is being applied can be seen in Figure 4. There we can see that even for larger graphs with several thousand nodes, the complexity is fairly constant. Furthermore, in none of our runs did the algorithm need more than 12 iterations to converge. We therefore assume that even for large values of n, the number of iterations needed is low and constant and would not add to the complexity of the overall algorithm.

8.5 Absolute error accuracy of approximate algorithm

Image size		10×10	50×50	100×100	200×200
CVX		0	0	0	0
Component-wise Alg.		0.30	0.80	0.73	1.0
Path Alg.	$K = 1$	0.31	0.81	0.73	1.0
	$K = 2$	0.26	0.75	0.73	1.0
	$K = 5$	0.22	0.74	0.73	0.90
	$K = 10$	0.15	0.68	0.73	0.90
	$K = 50$	0	0.44	0.22	0.52
	$K = 100$	0	0.35	0.18	0.43
	$K = 500$	0	0.025	0.047	0.14
	$K = 1000$	0	0	0.013	0.12
	$K = 2000$	0	0	0.00017	0.050
	$K = 5000$	0	0	0.00017	0.021
	exact	0	0	0	-

Table 4: Absolute error accuracy comparison for the 2-dimensional FLSA. The accuracy of the approximate version of the path algorithm and the component-wise algorithm are compared to the exact solution using the supremum norm. The largest error of the 50 values of λ_2 is reported. The results are averaged over 10 runs for the 10×10 and 50×50 dataset and 4 simulation runs for the rest.

9 One dimensional FLSA

9.1 The algorithm

In this subsection we will give a derivation of the 1-dimensional FLSA algorithm when we start at $\lambda_2 = 0$. For starting at $\lambda_2 = \infty$, a version is given at the end of this section. Calculating the path starting at 0 is very simple as groups of variables are only fused and never split. Due to the simpler structure, the definition of *fused* sets can also be simplified:

Definition 3. *Let F_i, $i = 1, \ldots, n_F(\lambda_2)$ be the sets of coefficients at λ_2 that are considered to be fused where $n_F(\lambda_2)$ is the number of such sets. Then every set F_i is of the form $F_i = \{k | l_i \leq k \leq u_i\}$ and the following statements hold:*

- $\cup_{i=1}^{n_F(\lambda_2)} F_i = \{1, \ldots, n\}$
- $F_i \cap F_j = \emptyset \quad for \quad i \neq j$
- *Assuming the F_i are ordered, for every $k, l \in F_i$ we have $\beta_k(\lambda_2) = \beta_l(\lambda_2)$ and for $k \in F_i, l \in F_{i+1}$ it holds that $\hat{\beta}_k(\lambda_2) \neq \hat{\beta}_l(\lambda_2)$.*

For notational convenience, write $\hat{\beta}_{F_i}(\lambda_2)$ for any $\hat{\beta}_k(\lambda_2)$ with $k \in F_i$ and also suppress the dependency of F_i onto λ_2. Using this definition of fused sets, let us now turn to the algorithm.

As already stated in the main article in Theorem 3, variables that have been fused at some value of λ_2^0 will remain fused for all $\lambda_2 \geq \lambda_2^0$. Using this theorem, the algorithm is just a specialization of Algorithm 1. First, we look at $\partial\hat{\beta}_k(\lambda_2)/\partial\lambda_2$. Inserting a graph $\mathcal{G}$ that is one-dimensional into Equation (7) we get

$$\begin{aligned}\frac{\partial\hat{\beta}_{F_i}}{\partial\lambda_2}(\lambda_2) = &- \frac{1}{|F_i(\lambda_2)|}(\text{sign}(\hat{\beta}_{F_i}(\lambda_2) - \hat{\beta}_{F_{i-1}}(\lambda_2))+ \\ &\text{sign}(\hat{\beta}_{F_i}(\lambda_2) - \hat{\beta}_{F_{i+1}}(\lambda_2))) \quad \text{for} \quad i = 1, \ldots, n_F(\lambda_2)\end{aligned}$$

where we set $\text{sign}(\hat{\beta}_{F_1}(\lambda_2) - \hat{\beta}_{F_0}(\lambda_2)) = 0$ and $\text{sign}(\hat{\beta}_{F_{n_F(\lambda_2)}}(\lambda_2) - \hat{\beta}_{F_{n_F(\lambda_2)+1}}(\lambda_2)) = 0$ for notational convenience as F_0 and $F_{n_F(\lambda_2)+1}$ do not exist.

As we have mentioned above, sets cannot break up for increasing values of λ_2, therefore the splitting time v is always infinite. It is only necessary to evaluate the next hitting time, which is particularly simple, as due to the one-dimensional graph, any set $F_i(\lambda_2)$ can only be fused with its left or right neighbor. Then, the hitting time for sets $F_i(\lambda_2)$ and $F_{i+1}(\lambda_2)$ is

$$h_{i,i+1}(\lambda_2) = \frac{\hat{\beta}_{F_i}(\lambda_2) - \hat{\beta}_{F_{i+1}}(\lambda_2)}{\frac{\partial\hat{\beta}_{F_{i+1}}(\lambda_2)}{\partial\lambda_2} - \frac{\partial\hat{\beta}_{F_i}(\lambda_2)}{\partial\lambda_2}} + \lambda_2 \quad \text{for} \quad i = 1, \ldots, n_F(\lambda_2) - 1.$$

Using this, we define the hitting time of any two sets exactly as before as

$$h(\lambda_2) = \min_{h_{i,i+1} > \lambda_2} h_{i,i+1}(\lambda_2)$$

and we fuse the two sets that hit each other. The slopes of the newly fused set needs to be calculated and this procedure is being repeated until there is only one set left. The resulting algorithm can be seen in Algorithm 2 in the main article.

9.2 Computational complexity

The computational complexity of the algorithm is $O(n\log(n))$, which we will see in the following paragraphs. First note that the initialization step takes $2n+2$ operations. However, most of the computations are performed for calculating the next hitting time $h(\lambda_2)$. When $\lambda_2 = 0$, the derivative $\partial\hat{\beta}_{F_i}(\lambda_2)/\partial\lambda_2$ has to be computed for all n groups and the smallest hitting time identified. This takes on the order $O(n)$ operations. However, in subsequent steps, $h_{i,i+1}(\lambda_2)$ remain the same except if either $F_i(\lambda_2)$ or $F_{i+1}(\lambda_2)$ was fused with its neighbor in the last iterations. This way, in each iteration only 2 values of $h_{i,i+1}(\lambda_2)$ have to be

updated. Finding the smallest value of $h_{i,i+1}(\lambda_2)$ requires operations on the order $O(\log(n))$ if an efficient data structure is used to save the $h_{i,i+1}(\lambda_2)$ (e.g. a binary tree). The same is true for the updates of the $\hat{\beta}_{F_i}(\lambda_2)$. Only the $\hat{\beta}_{F_i}(\lambda_2)$ of the groups that have just been fused have to be updated. For the other groups $\partial\hat{\beta}_{F_i}(\lambda_2)/\partial\lambda_2$ stays the same, so we can interpolate $\hat{\beta}_{F_i}(\lambda_2)$ when we need it based on λ_2^0 when this group was created as well as $\hat{\beta}_{F_i}(\lambda_2^0)$ and its derivative. So this also only involves a constant number of operations per iteration. As with every iteration, the number of sets $n_F(\lambda_2)$ decreases by one, there are at most $n-1$ iterations. Therefore, the entire solution path can be obtained with a computational complexity on the order $O(n\log(n))$.

9.3 Storing the solution path efficiently

One possibility to store and retrieve the optimal coefficients for a value of λ_2 would be to store the whole coefficient vector $\boldsymbol{\beta}(\lambda_2)$ for every value of λ_2 for which 2 sets are fused and interpolate linearly in between. However, then it would be necessary to store n^2 entries, which would be impossible for large values of n. However, when done efficiently, it is possible to store the entire solution with a memory requirement that only grows linearly in n. For this, the result is stored in the form of a binary tree. Every node contains a value for λ_2 at which this node (corresponding to a set of fused variables) becomes active and the correct value of $\hat{\beta}_{F_i}(\lambda_2)$ for this same value λ_2.

We start by creating n unconnected nodes corresponding to the n coefficients for $\lambda_2 = 0$ and $\hat{\beta}_k(0) = y_k$. Then when two sets of coefficients fuse, add a new node that is the parent of the two nodes corresponding to the 2 groups that were just fused. In the new node, store the value of λ_2 as well as the updated value of the coefficient $\hat{\beta}_{F_i}(\lambda_2)$. At the end of the algorithm, we will have one binary tree (for a small example see Figure 5). In order to retrieve one value of the solution at $\tilde{\lambda}_2$ for coefficient β_k, first start at the leaf node of coefficient β_k. Climb up the tree until λ_2 of the next parent node is larger than $\tilde{\lambda}_2$. The correct solution is then a linear interpolation between the $\hat{\beta}_{F_i}(\lambda_2)$ values stored in the current node and the parent node. Using this method, the complexity for looking the whole vector of solutions $\hat{\boldsymbol{\beta}}(\lambda_2)$ for a particular value of λ_2 is then also $O(n\log n)$. Therefore, running the path algorithm and getting a solution vector has a total complexity of $O(n\log n)$. We will also be able to see this in the next section when we compare the speed of this algorithm to other existing methods for this problem.

9.4 Alternative proof of Theorem 3

An important prerequisite for the one-dimensional algorithm is the result of Theorem 3 in the article. A proof has already been given in Friedman et al. [2007] and here we want to

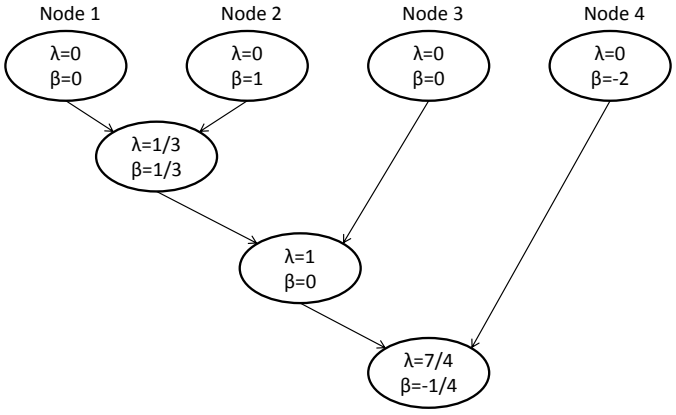

Figure 5: Example of a tree for storing the solution path of the one-dimensional FLSA.

give an alternative version. It is based on the results of the General Fused Lasso Signal Approximator in Section 2.

Theorem 3. *For the one-dimensional FLSA assume that for some* $\lambda_2^0 > 0$ *and* $\lambda_1 = 0$*, we have that* $\hat{\beta}_k(\lambda_2^0) = \hat{\beta}_{k+1}(\lambda_2^0)$ *for some* $1 \leq k \leq N-1$*. Then, for any* $\lambda_2 > \lambda_2^0$ *we have that* $\hat{\beta}_k(\lambda_2) = \hat{\beta}_{k+1}(\lambda_2)$*.*

Proof. First note that in Section 2.3.4, we described the maximum flow problem that we have to solve and under which conditions it is necessary to break up the set of fused variables. In the case of the one-dimensional FLSA, the maximum flow graph is especially simple. Assume that an arbitrary set of fused variables $|F|$ is $\{k_0, k_0+1, \ldots, k_0+|F|-1\}$, which has the edges $\{(k_0, k_0+1), \ldots, (k_0+|F|-2, k_0+|F|-1)\}$. For the capacities on these edges, it is sufficient to note that they are by definition always ≥ 1. Now it only remains to specify which edges are connected to the source node r and the sink node s and which capacities they have. Essentially, at λ_2^0, there are 4 possible situations:

Case 1: $\hat{\beta}_{\mathbf{k_0-1}}(\lambda_2^0), \hat{\beta}_{\mathbf{k_0+|F|}}(\lambda_2^0) > \hat{\beta}_{\mathbf{F}}(\lambda_2^0)$**:** In this case we have that $\frac{\partial \hat{\beta}_F(\lambda_2^0)}{\partial \lambda_2} = \frac{2}{|F|}$. Then we have that k_0 and $k_0+|F|-1$ are connected to the source r and all other nodes are connected to the sink s. For the capacities from the source, we have $c_{rl} = 1 - \frac{2}{|F|} \leq 1$ and for the capacities to the sink it holds that $c_{ks} = \frac{2}{|F|} \leq 1$. As the capacities on the inner nodes is at least 1 as stated above, it is easy to see that for the maximum flow we always have $f_{rl} = c_{rl}$ for $l = k_0, k_0+|F|-1$ and therefore the group will never be split.

Case 2: $\hat{\beta}_{\mathbf{k_0+1}}(\lambda_2^0) > \hat{\beta}_{\mathbf{F}}(\lambda_2^0) > \hat{\beta}_{\mathbf{k_0+|F|}}(\lambda_2^0)$**:** Here, we have that $\frac{\partial \hat{\beta}_F}{\partial \lambda_2}(\lambda_2^0) = 0$ and node k_0 is connected to the source r whereas $k_0+|F|-1$ is connected to the sink s. All other nodes are not connected to either the source or the sink. For the capacities in this case we have $c_{rk_0} = c_{k_0+|F|-1,s} = 1$ and therefore the maximum flow is always

$f_{rk_0} = f_{k_0,k_0+1} = \cdots = f_{k_0+|F|-2,k_0+|F|-1} = f_{k_0+|F|-1,s} = 1$ and therefore, the group will never be split.

Case 3: $\hat{\beta}_{\mathbf{k_0+1}}(\lambda_2^0) < \hat{\beta}_{\mathbf{F}}(\lambda_2^0) < \hat{\beta}_{\mathbf{k_0+|F|}}(\lambda_2^0)$**:** Similar to case 2.

Case 4: $\hat{\beta}_{\mathbf{k_0-1}}(\lambda_2^0), \hat{\beta}_{\mathbf{k_0+|F|}}(\lambda_2^0) < \hat{\beta}_{\mathbf{F}}(\lambda_2^0)$**:** Similar to case 1.

In these cases we assumed that $k_0 \neq 1$ and $k_0 + |F| - 1 \neq N$, i.e. that the set of fused variables is not at the boundary of the graph. However, it is easy to see that in this case the set will also not break apart for increasing λ_2. Therefore, the proposition holds. □

9.5 Algorithm starting at $\lambda_2 = \infty$

Here we will present the algorithm that solves the one-dimensional FLSA where we start at $\lambda_2 = \infty$ and calculate the path while decreasing λ_2 down to 0. As before, the function we are minimizing is

$$L(\mathbf{y}, \boldsymbol{\beta}) = \frac{1}{2}\sum_{k=1}^{n}(y_k - \beta_k)^2 + \lambda_2 \sum_{k=1}^{n-1}|\beta_k - \beta_{k+1}|$$

where we leave out the term $\lambda_1 \sum_{k=1}^{n} |\beta_k|$ as we can use soft-thresholding for it as shown in Theorem 1. In this algorithm, we will also use Theorem 3 (proof above) which states that that groups of "fused" variables do not break up for increasing values of λ_2. As we will decrease λ_2 in this procedure, this is equivalent to the statement that groups that break up for decreasing λ_2 will never merge again.

As before, we take derivatives w.r.t β_k (and subgradients where those don't exist) and get:

$$\frac{\partial L(\mathbf{y}, \boldsymbol{\beta}, \lambda_2)}{\partial \beta_k} = \beta_k - y_k + \lambda_2(-t_{k-1,k} + t_{k,k+1}) \tag{13}$$

where for notational convenience $t_{0,1} = t_{n,n+1} = 0$. Also $t_{k,k+1} \in [-1, 1]$ if $\beta_k = \beta_{k+1}$ and $t_{k,k+1} = \text{sign}(\beta_k - \beta_{k+1})$. For shorthand we write $\tau_{k,k+1} = \lambda_2 t_{k,k+1}$ and then require that $\tau_{k,k+1} \in [-\lambda_2, \lambda_2]$ if $\beta_k \neq \beta_{k+1}$.

9.5.1 Starting point

For large values of λ_2, we just have one group $F(\infty) = \{1, \ldots, n\}$ and it is easy to verify that the value of the coefficients in this group is $\hat{\beta}_F(\infty) = \frac{1}{n}\sum_{k=1}^{n} y_k$, the mean of all input values. In order to start the algorithm, we also have to specify the values for $\tau_{k,k+1}$. In order

to satisfy the equations above with $\partial L/\partial\beta_k$ for $\hat{\beta}_k = \hat{\beta}_F = \overline{y}$, we pick

$$\hat{\tau}_{12}(\infty) = y_1 - \overline{y}$$
$$\hat{\tau}_{k,k+1}(\infty) = y_k - \overline{y} + \hat{\tau}_{k-1,k}$$

for $k > 1$. It only remains to check that this also holds for $\hat{\tau}_{n-1,n}(\infty)$. By our definition, we have

$$\hat{\tau}_{n-1,n}(\infty) = y_{n-1} - \overline{y} + \hat{\tau}_{n-2,n-1} = \sum_{k=1}^{n-1} y_k - (n-1)\overline{y}$$

which we can show by induction and inserting this into $\partial L/\partial\beta_k$ yields the required

$$\hat{\beta}_n(\infty) - y_n - \hat{\tau}_{n-1,n}(\infty) = n\overline{y} - \sum_{k=1}^{n} y_k = 0.$$

Using this starting value, we have to split a group at position $(k, k+1)$ as soon as $\hat{\tau}_{k,k+1}(\lambda_2) \in [-\lambda_2, \lambda_2]$ is not satisfied anymore.

9.5.2 Splitting sets

As we now know how to start the algorithm, assume that we are at step K, i.e. we have K groups. Therefore assume that for some value of λ_2, we have $K = n_F(\lambda_2)$ groups that we call $F_1(\lambda_2), \ldots, F_K(\lambda_2)$ and each group is an interval $F_i(\lambda_2) = [l_i(\lambda_2), r_i(\lambda_2)]$. The coefficient of the group is $\hat{\beta}_{F_i}(\lambda_2)$ and we also know the values $\hat{\tau}_{k,k+1}(\lambda_2)$. We now have to calculate the derivatives of $\hat{\beta}_{F_i}(\lambda_2)$ and $\hat{\tau}_{k,k+1}(\lambda_2)$ w.r.t. λ_2. Taking the grouping in the loss function into account, we have

$$L(\mathbf{y}, \boldsymbol{\beta}, \lambda_2) = \frac{1}{2}\sum_{i=1}^{K}\left(\left(\sum_{k=l_i}^{r_i} y_k\right) - |F_i|\beta_{F_i}(\lambda_2)\right)^2 + \lambda_2 \sum_{i=1}^{K-1} |\beta_{F_i}(\lambda_2) - \beta_{F_{i+1}}(\lambda_2)|$$

where we can just take the derivatives as $\hat{\beta}_{F_i}(\lambda_2) \neq \hat{\beta}_{F_{i+1}}(\lambda_2)$ for all i. Setting them to 0, and solving for the solutions $\hat{\boldsymbol{\beta}}(\lambda_2)$ gives

$$\begin{aligned}
\frac{\partial L(y,\beta)}{\partial \beta_{F_1}} &= |F_1|(\lambda_2)\hat{\beta}_{F_1}(\lambda_2) - \left(\sum_{k=l_1}^{r_1} y_k\right) + \lambda_2 \text{sign}(\hat{\beta}_{F_1}(\lambda_2) - \hat{\beta}_{F_2}(\lambda_2)) = 0 \\
\frac{\partial L(y,\beta)}{\partial \beta_{F_i}} &= |F_i(\lambda_2)|\hat{\beta}_{F_i}(\lambda_2) - \left(\sum_{k=l_i(\lambda_2)}^{r_i(\lambda_2)} y_k\right) \\
&\quad + \lambda_2 \left(\text{sign}(\hat{\beta}_{F_i}(\lambda_2) - \hat{\beta}_{F_{i+1}}(\lambda_2)) + \text{sign}(\hat{\beta}_{F_i}(\lambda_2) - \hat{\beta}_{F_{i-1}}(\lambda_2))\right) = 0 \\
\frac{\partial L(y,\beta)}{\partial \beta_{F_K}} &= |F_K|(\lambda_2)\hat{\beta}_{F_K}(\lambda_2) - \left(\sum_{k=l_K(\lambda_2)}^{r_K(\lambda_2)} y_k\right) + \lambda_2 \text{sign}(\hat{\beta}_{F_K}(\lambda_2) - \hat{\beta}_{F_{K-1}}(\lambda_2)) = 0
\end{aligned}$$

where $1 < i < K$. Taking the derivatives w.r.t. λ_2 then yields

$$\begin{aligned}
\frac{\partial \beta_{F_1}}{\partial \lambda_2} &= \frac{\text{sign}(\hat{\beta}_{F_1}(\lambda_2) - \hat{\beta}_{F_2}(\lambda_2))}{|F_1(\lambda_2)|} \\
\frac{\partial \beta_{F_i}}{\partial \lambda_2} &= \frac{\text{sign}(\hat{\beta}_{F_i(\lambda_2)} - \hat{\beta}_{F_{i+1}}(\lambda_2)) + \text{sign}(\hat{\beta_{F_i}} - \hat{\beta}_{F_{i-1}})}{|F_i(\lambda_2)|} \\
\frac{\partial \hat{\beta}_{F_K}(\lambda_2)}{\partial \lambda_2} &= \frac{\text{sign}(\hat{\beta}_{F_K}(\lambda_2) - \hat{\beta}_{F_{K-1}}(\lambda_2))}{|F_K(\lambda_2)|}.
\end{aligned}$$

From setting the derivative of the subgradients in Equation (13) to 0 and taking the derivative w.r.t λ_2 it follows:

$$\frac{\partial \hat{\beta}_k}{\partial \lambda_2}(\lambda_2) + \frac{\partial \hat{\tau}_{k,k+1} - \partial \hat{\tau}_{k-1,k}}{\partial \lambda_2}(\lambda_2) = 0$$

for each $1 \leq k \leq K$. We only have to solve for $\frac{\partial \hat{\tau}_{k,k+1}}{\partial \lambda_2}(\lambda_2)$ for which $k, k+1 \in F_i$ for some i, i.e. both nodes are in the same group. For $k \in F_i(\lambda_2), k+1 \in F_{i+1}(\lambda_2)$ we know by definition that $\hat{\tau}_{k,k+1}(\lambda_2) = \lambda_2 \text{sign}(\hat{\beta}_{k+1}(\lambda_2) - \hat{\beta}_k(\lambda_2))$. As there are only few cases, we can enumerate them

1. $i = 1$, $\hat{\beta}_{F_2}(\lambda_2) > \hat{\beta}_{F_1}(\lambda_2)$: Then $\frac{\partial \hat{\beta}_{F_1}}{\partial \lambda_2}(\lambda_2) = \frac{1}{|F_1(\lambda_2)|}$. Furthermore $\frac{\partial \hat{\tau}_{k,k+1}}{\partial \lambda_2}(\lambda_2) = -\frac{k}{|F_1(\lambda_2)|}$ for $1 \leq k \leq |F_1(\lambda_2)|$.

2. $1 < i < K$, $\hat{\beta}_{F_i+1}(\lambda_2) > \hat{\beta}_{F_i}(\lambda_2)$, $\hat{\beta}_{F_{i-1}}(\lambda_2) > \hat{\beta}_{F_i}(\lambda_2)$: Here $\frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2) = \frac{2}{|F_i(\lambda_2)|}$. Also $\frac{\partial \hat{\tau}_{k,k+1}}{\partial \lambda_2}(\lambda_2) = 1 - \frac{2(k - l_i(\lambda_2)+1)}{|F_i(\lambda_2)|}$ for $l_i(\lambda_2) \leq k \leq r_i(\lambda_2)$.

3. $1 < i < K$, $\hat{\beta}_{F_{i+1}}(\lambda_2) > \hat{\beta}_{F_i}(\lambda_2)$, $\hat{\beta}_{F_{i-1}}(\lambda_2) < \hat{\beta}_{F_i}(\lambda_2)$: Now $\frac{\partial \hat{\beta}_{F_i}}{\partial \lambda_2}(\lambda_2) = 0$. Also $\frac{\partial \hat{\tau}_{i,i+1}}{\partial \lambda_2}(\lambda_2) = -1$.

The other cases can be obtained by simple symmetry from the ones mentioned above.

Based on these calculations, for each group we can find the value of λ_2 for which it breaks apart, i.e. for which for a decreasing value of λ_2 we have either $\hat{\tau}_{k,k+1}(\lambda_2) = \lambda_2$ or $\hat{\tau}_{k,k+1}(\lambda_2) = -\lambda_2$. The breakup time at position $(k, k+1)$ is

$$b_{k,k+1}(\lambda_2) = \max\{b^+_{k,k+1}(\lambda_2), b^-_{k,k+1}(\lambda_2)\}$$

where $b^+_{k,k+1}(\lambda_2)$ occurs when $\hat{\tau}_{k,k+1}(\lambda_2)$ hits λ_2 and $b^-_{k,k+1}(\lambda_2)$ when it hits $-\lambda_2$. Specifically

$$b^+_{k,k+1}(\lambda_2) = \begin{cases} \lambda_2 + \frac{\lambda_2 - \hat{\tau}_{k,k+1}(\lambda_2)}{\frac{\partial \hat{\tau}_{k,k+1}}{\partial \lambda_2} - 1} & \text{if } \frac{\partial \hat{\tau}_{k,k+1}}{\partial \lambda_2} > 1 \\ 0 & \text{otherwise} \end{cases}$$

and

$$b^-_{k,k+1}(\lambda_2) = \begin{cases} \lambda_2 + \frac{-\lambda_2 - \hat{\tau}_{k,k+1}(\lambda_2)}{\frac{\partial \hat{\tau}_{k,k+1}}{\partial \lambda_2} + 1} & \text{if } \frac{\partial \hat{\tau}_{k,k+1}}{\partial \lambda_2} < -1 \\ 0 & \text{otherwise.} \end{cases}$$

Therefore the breakup time for group i is

$$b_i(\lambda_2) = \max_{l_i \le k, k+1 \le r_i} b_{k,k+1}(\lambda_2).$$

Putting all this together, we get Algorithm 4.

Algorithm 4: Backwards FLSA path algorithm

initialize
- $\lambda_2 = \infty$;
- $\beta_k = \overline{y}$ for $k = 1, \ldots, n$;
- $\tau_{12} = y_1 - \overline{y}$;
- $\tau_{k,k+1} = y_k - \overline{y} + \tau_{k-1,k}$ for $k = 2, \ldots n-1$;
- $F_1 = \{1, \ldots, n\}$;
- $n_F = 1$

end

while $n_F < n$ **do**
- For all n_F groups check for which λ_2 it breaks up;
- Pick the group with the largest breakup time b_i;
- Set $\lambda_2 := \max b_i$;
- Break the group in two;
- Update β and τ;
- $n_F := n_F + 1$;

end

Next, we will discuss the complexity of this procedure, especially when not calculating the whole path but only up to a certain number of fused groups of variables.

9.5.3 Complexity

When calculating the computational complexity, we assume that we only let the algorithm run until we have K groups for which we output the solutions for a total number of L groups. Initializing the algorithm has a complexity of n. For every iteration, searching for the currently largest break time is of complexity $\log(n_F)$ (as only one gets updated every iteration). Outputting the results has a complexity of nL. Together, the total complexity is $O(K \log(n) + nL)$.

References

Dimitri P. Bertsekas. *Nonlinear Programming.* Athena Scientific, 1999.

Jerome Friedman, Trevor Hastie, Holger Hoefling, and Robert Tibshirani. Pathwise coordinate optimization. *Annals of Applied Statistics*, 2(1):302–332, 2007.

D.D. Sleator and R.E. Tarjan. A data structure for dynamic trees. *Journal of Journal of Computer and System Sciences*, 24:362–391, 1983.