A General Framework for Fast Stagewise Algorithms

Ryan J. Tibshirani
Carnegie Mellon University
ryantibs@stat.cmu.edu

Abstract

Forward stagewise regression follows a very simple strategy for constructing a sequence of sparse regression estimates: it starts with all coefficients equal to zero, and iteratively updates the coefficient (by a small amount ϵ) of the variable that achieves the maximal absolute inner product with the current residual. This procedure has an interesting connection to the lasso: under some conditions, it is known that the sequence of forward stagewise estimates exactly coincides with the lasso path, as the step size ϵ goes to zero. Furthermore, essentially the same equivalence holds outside of least squares regression, with the minimization of a differentiable convex loss function subject to an ℓ_1 norm constraint (the stagewise algorithm now updates the coefficient corresponding to the maximal absolute component of the gradient).

Even when they do not match their ℓ_1-constrained analogues, stagewise estimates provide a useful approximation, and are computationally appealing. Their success in sparse modeling motivates the question: can a simple, effective strategy like forward stagewise be applied more broadly in other regularization settings, beyond the ℓ_1 norm and sparsity? The current paper is an attempt to do just this. We present a general framework for stagewise estimation, which yields fast algorithms for problems such as group-structured learning, matrix completion, image denoising, and more.

Keywords: *forward stagewise regression, lasso, ϵ-boosting, regularization paths*

1 Introduction

In a regression setting, let $y \in \mathbb{R}^n$ denote an outcome vector and $X \in \mathbb{R}^{n \times p}$ a matrix of predictor variables, with columns $X_1, \ldots X_p \in \mathbb{R}^n$. For modeling y as a linear function of X, we begin by considering (among the many possible candidates for sparse estimation tools) a simple method: *forward stagewise regression.* In words, forward stagewise regression produces a sequence of coefficient estimates $\beta^{(k)}$, $k = 0, 1, 2, \ldots$, by iteratively decreasing the maximal absolute inner product of a variable with the current residual, each time by only a small amount. A more precise description of the algorithm is as follows.

Algorithm 1 (**Forward stagewise regression**)**.**

Fix $\epsilon > 0$, initialize $\beta^{(0)} = 0$, and repeat for $k = 1, 2, 3, \ldots$,

$$\beta^{(k)} = \beta^{(k-1)} + \epsilon \cdot \mathrm{sign}\big(X_i^T (y - X\beta^{(k-1)})\big) \cdot e_i, \tag{1}$$

$$\textit{where } i \in \operatorname*{argmax}_{j=1,\ldots p} |X_j^T (y - X\beta^{(k-1)})|. \tag{2}$$

In the above, $\epsilon > 0$ is a small fixed constant (e.g., $\epsilon = 0.01$), commonly referred to as the step size or learning rate; e_i denotes the ith standard basis vector in $\mathbb{R}^p$; and the element notation in (2) emphasizes that the maximizing index i need not be unique. The basic idea behind the forward stagewise updates (1), (2) is highly intuitive: at each iteration we greedily select the variable i that has the largest absolute inner product (or correlation, for standardized variables) with the residual,

and we add $s_i\epsilon$ to its coefficient, where s_i is the sign of this inner product. Accordingly, the fitted values undergo the update:

$$X\beta^{(k)} = X\beta^{(k-1)} + \epsilon \cdot s_i X_i.$$

Such greediness, in selecting variable i, is counterbalanced by the small step size $\epsilon > 0$; instead of increasing the coefficient of X_i by a (possibly) large amount in the fitted model, forward stagewise only increases it by ϵ, which "slows down" the learning process. As a result, it typically requires many iterations to produce estimates of reasonable interest with forward stagewise regression, e.g., it could easily take thousands of iterations to reach a model with only tens of active variables (we use "active" here to refer to variables that are assigned nonzero coefficients). See the left panel of Figure 1 for a small example.

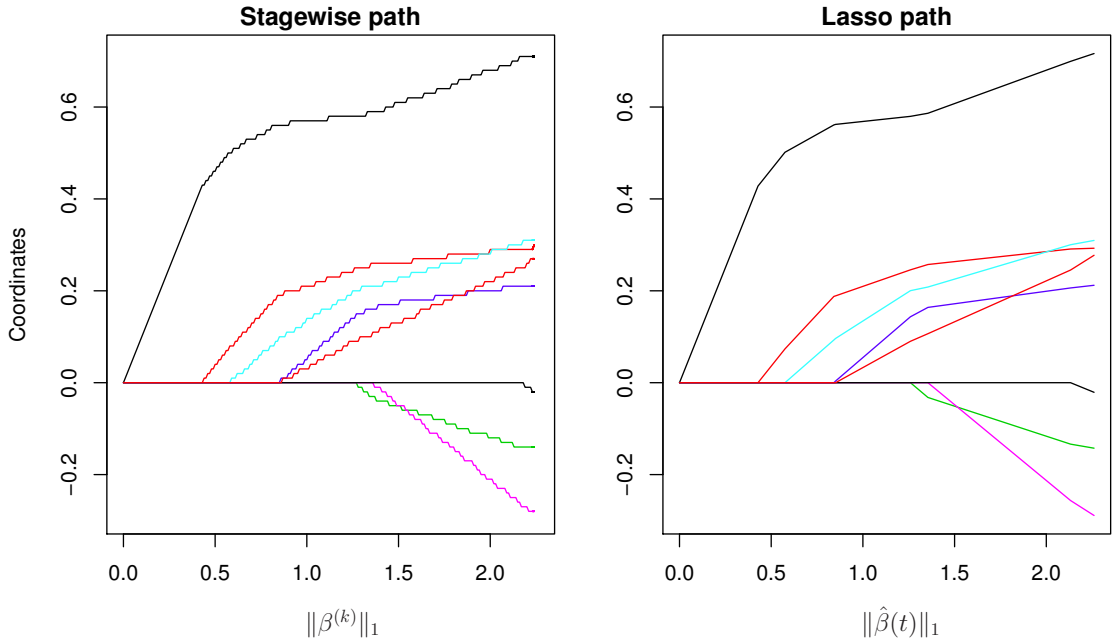


Figure 1: *A simple example using the prostate cancer data from Hastie et al. (2009), where the log PSA score of $n = 67$ men with prostate cancer is modeled as a linear function of $p = 8$ biological predictors. The left panel shows the forward stagewise regression estimates $\beta^{(k)} \in \mathbb{R}^8$, $k = 1, 2, 3, \ldots$, with the 8 coordinates plotted in different colors. The stagewise algorithm was run with $\epsilon = 0.01$ for 250 iterations, and the x-axis here gives the ℓ_1 norm of the estimates across iterations. The right panel shows the lasso solution path, also parametrized by the ℓ_1 norm of the estimate. The similarity between the stagewise and lasso paths is visually striking; for small enough ϵ, they appear identical. This is not a coincidence and has been rigorously studied by Efron et al. (2004), and other authors; in Section 2.1 we provide an intuitive explanation for this phenomenon.*

This "slow learning" property is a key difference between forward stagewise regression and the closely-named *forward stepwise regression* procedure: at each iteration, the latter algorithm chooses a variable in a similar manner to that in (2)[1], but once it does so, it updates the fitted model by regressing y on all variables selected thus far. While both are greedy algorithms, the stepwise procedure is much greedier; after k iterations, it produces a model with exactly k active variables.

[1]If A denotes the active set at the end of iteration $k-1$, then at iteration k forward stepwise chooses the variable i such that the sum of squared errors from regressing y onto the variables in $A \cup \{i\}$ is smallest. This is equivalent to choosing i such that $|\widetilde{X}_i^T(y - X\beta^{(k-1)})|$ is largest, where $\beta^{(k-1)}$ denote the coefficients from regressing y on the variables in A, and $\widetilde{X}_i$ is the residual from regressing X_i on the variables in A.

Forward stagewise and forward stepwise are old techniques (some classic references for stepwise regression methods are Efroymson (1966) and Draper & Smith (1966), but there could have been earlier relevant work). According to Hastie et al. (2009), forward stagewise was historically dismissed by statisticians as being "inefficient" and hence less useful than methods like forward or backward stepwise. This is perhaps understandable, if we keep in mind the limited computational resources of the time. From a modern perspective, however, we now appreciate that "slow learning" is a form of regularization and can present considerable benefits in terms of the generalization error of the fitted models—this is seen not only in regression, but across variety of settings. Furthermore, by modern standards, forward stagewise is computationally cheap: to trace out a path of regularized estimates, we repeat very simple iterations, each one requiring (at most) p inner products, computations that could be trivially parallelized.

The revival of interest in stagewise regression began with the work of Efron et al. (2004), where the authors derived a surprising connection between the sequence of forward stagewise estimates and the solution path of the *lasso* (Tibshirani 1996),

$$\hat{\beta}(t) = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \ \frac{1}{2}\|y - X\beta\|_2^2 \ \text{ subject to } \ \|\beta\|_1 \leq t, \tag{3}$$

over the regularization parameter $t \geq 0$. The relationship between stagewise and the lasso will be reviewed in Section 2.1 in detail, but the two panels in Figure 1 tell the essence of the story. The stagewise paths, on the left, appear to be jagged versions of their lasso counterparts, on the right. Indeed, as the step size ϵ is made smaller, this jaggedness becomes less noticeable, and eventually the two sets of paths appear exactly the same. This is not a coincidence, and under some conditions (on the problem instance in consideration), it is known that the stagewise path converges to the lasso path, as $\epsilon \to 0$. Interestingly, when these conditions do not hold, stagewise estimates can deviate substantially from lasso solutions, and yet in such situations the former estimates can still perform competitively with the latter, say, in terms of test error (or really any other standard error metric). This is an important point, and it supports the use of stagewise regression as a general tool for regularized estimation.

1.1 Summary of our contributions

This paper departs from the lasso setting and considers the generic convex problem

$$\hat{x}(t) \in \underset{x \in \mathbb{R}^n}{\operatorname{argmin}} \ f(x) \ \text{ subject to } \ g(x) \leq t, \tag{4}$$

where $f, g : \mathbb{R}^n \to \mathbb{R}$ are convex functions, and f is differentiable. Motivated by forward stagewise regression and its connection to the lasso, our main contribution is the following *general stagewise algorithm* for producing an approximate solution path of (4), as the regularization parameter t varies over $[t_0, \infty)$.

Algorithm 2 (**General stagewise procedure**)**.**

Fix $\epsilon > 0$ and $t_0 \in \mathbb{R}$. Initialize $x^{(0)} = \hat{x}(t_0)$, a solution in (4) at $t = t_0$. Repeat, for $k = 1, 2, 3, \ldots$,

$$x^{(k)} = x^{(k-1)} + \Delta, \tag{5}$$

$$\textit{where } \Delta \in \underset{z \in \mathbb{R}^n}{\operatorname{argmin}} \ \langle \nabla f(x^{(k-1)}), z \rangle \ \text{ subject to } \ g(z) \leq \epsilon. \tag{6}$$

The intuition behind the general stagewise algorithm can be seen right away: at each iteration, we update the current iterate in a direction that minimizes the inner product with the gradient of f (evaluated at the current iterate), but simultaneously restrict this direction to be small under g. By applying these updates repeatedly, we implicitly adjust the trade-off between minimizing f and

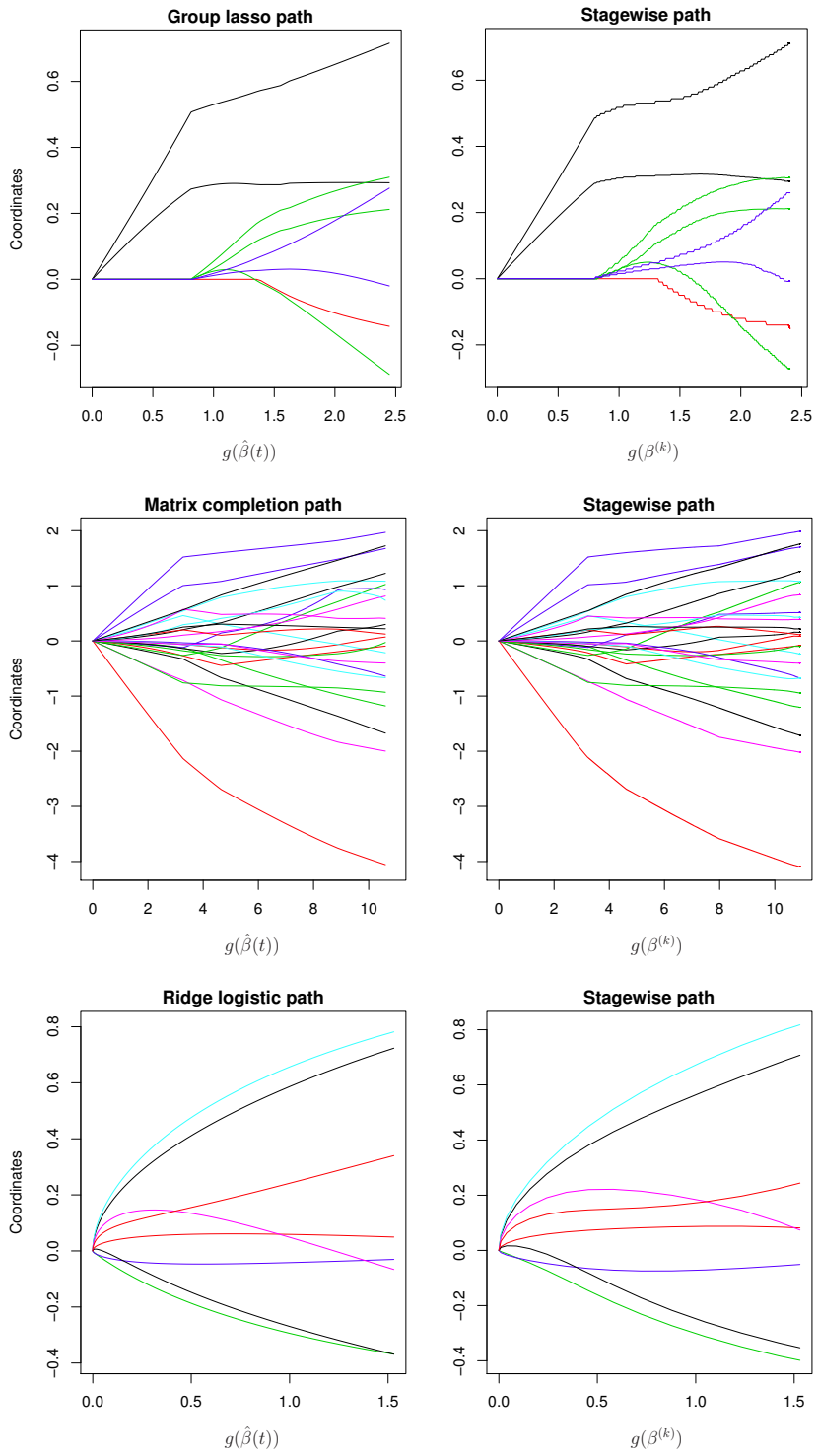


Figure 2: *Examples comparing the actual solution paths (left column) to the stagewise paths (right column) across various problem contexts, using the prostate cancer data set. The first row considers a group lasso model on the prostate data (where the groups were somewhat arbitrarily chosen based on the predictor types); the second row considers a matrix completion task, on a partially observed submatrix of the full predictor matrix; the third row considers a logistic regression model with ridge regularization (the outcome being the indicator of log PSA* > 1*). In each case, the stagewise estimates were very easy to compute; Sections 3.1, 3.3, and 3.4 discuss these problem settings in detail.*

g, and hence one can imagine that the kth iterate $x^{(k)}$ approximately solves (4) with $t = g(x^{(k)})$. In Figure 2, we show a few simple examples of the general stagewise paths implemented for various different choices of loss functions f and regularizing functions g.

In the next section, we develop further intuition and motivation for the general stagewise procedure, and we tie in forward stagewise regression as a special case. The rest of this article is then dedicated to the implementation and analysis of stagewise algorithms: Section 3 derives the specific form of the stagewise updates (5), (6) for various problem setups, Section 4 conducts large-scale empirical evaluations of stagewise estimates, Section 5 presents some theory on suboptimality, and Section 6 concludes with a discussion.

Throughout, our arguments and examples are centered around three points, summarized below.

1. *Simple, fast estimation procedures.* The general framework for stagewise estimation in Algorithm 2 leads to simple and efficient stagewise procedures for group-structured regularization problems (e.g., the group lasso, multitask learning), trace norm regularization problems (e.g., matrix completion), quadratic regularization problem problems (e.g., nonparametric smoothing), and (some) generalized lasso problems (e.g., image denoising). For such problems, the proposed stagewise procedures are often competitive with existing commonly-used algorithms in terms of efficiency, and are generally much simpler.

2. *Similar to actual solution paths, but more stable.* In many examples, the computed stagewise path is highly similar to the actual solution path of the corresponding convex regularization problem in (4)—typically, this happens when the components of the actual solution change "slowly" with the regularization parameter t. In many others, even though it shares gross characteristics of the actual solution path, the stagewise path is different—typically, this happens when the components of the actual solution change "rapidly" with t, and the stagewise component paths are much more stable.

3. *Competitive statistical performance.* Across essentially all cases, even those in which its constructed path is not close to the actual solution path, the stagewise algorithm performs favorably from a statistical point of view. That is, stagewise estimates are comparable to solutions in (4) with respect to relevant error metrics, across various problem settings. This suggests that stagewise estimates deserved to be studied on their own, regardless of their proximity to solutions in (4).

The third point above, on the favorable statistical properties of stagewise estimates, is based on empirical arguments, rather than theoretical ones. Statistical theory for stagewise estimates is an important topic for future work.

2 Properties of the general stagewise framework

2.1 Motivation: stagewise regression and the lasso

The lasso estimator is a popular tool for sparse estimation in the regression setting. Displayed in (3), we assume for simplicity that the lasso solution $\hat{\beta}(t)$ in (3) is unique, which holds under very weak conditions on X.[2] Recall that the parameter t controls the level of sparsity in the estimate $\hat{\beta}(t)$: when $t = 0$, we have $\hat{\beta}(0) = 0$, and as t increases, select components of $\hat{\beta}(t)$ become nonzero, corresponding to variables entering the lasso model (nonzero components of $\hat{\beta}(t)$ can also become zero, corresponding to variables leaving the model). The solution path $\hat{\beta}(t)$, $t \in [0, \infty)$ is continuous

[2]For example, it suffices to assume that X has columns in general position, see Tibshirani (2013). Note that here we are only claiming uniqueness for all parameter values $t < t^*$, where t^* is the smallest ℓ_1 norm of a least squares solution of y on X.

and piecewise linear as a function of t, and for a large enough value of t, the path culminates in a least squares estimate of y on X.

The right panel of Figure 1 shows an example of the lasso path, which, as we discussed earlier, appears quite similar to the stagewise path on the left. This is explained by the seminal work of Efron et al. (2004), who describe two algorithms (actually three, but the third is unimportant for our purposes): one for explicitly constructing the lasso path $\hat{\beta}(t)$ as a continuous, piecewise linear function of the regularization parameter $t \in [0, \infty)$, and another for computing the limiting stagewise regression paths as $\epsilon \to 0$. One of the (many) consequences of their work is the following: if each component of the lasso solution path $\hat{\beta}(t)$ is a monotone function of t, then these two algorithms coincide, and therefore so do the stagewise and lasso paths (in the limit as $\epsilon \to 0$). Note that the lasso paths for the data example in Figure 1 are indeed monotone, and hence the theory confirms the observed convergence of stagewise and lasso estimates in this example.

The lasso has undergone intense study as a regularized regression estimator, and its statistical properties (e.g., its generalization error, or its ability to detect a truly relevant set of variables) are more or less well-understood at this point. Many of these properties cast the lasso in a favorable light. Therefore, the equivalence between the (limiting) stagewise and lasso paths lends credibility to forward stagewise as a regularized regression procedure: for a small step size ϵ, we know that the forward stagewise estimates will be close to lasso estimates, at least when the individual coordinate paths are monotone. At a high level, it is actually somewhat remarkable that such a simple algorithm, Algorithm 1, can produce estimates that can stand alongside those defined by the (relatively) sophisticated optimization problem in (3). There are now several interesting points to raise.

- *The nonmonotone case.* In practice, the components of the lasso path are rarely monotone. How do the stagewise and lasso paths compare in such cases? A precise theoretical answer is not known, but empirically, these paths can be quite different. In particular, for problems in which the predictors $X_1, \ldots X_p$ are correlated, the lasso coordinate paths can be very wiggly (as variables can enter and leave the model repeatedly), while the stagewise paths are often very stable; see, e.g., Hastie et al. (2007). In support of these empirical findings, the latter authors derived a local characterization of the lasso and forward stagewise paths: they show that at any point along the path, the lasso estimate decreases the sum of squares loss function at an optimal rate with respect to the increase in ℓ_1 norm, and the (limiting) forward stagewise estimate decreases the loss function at an optimal rate with respect to the increase in ℓ_1 arc length. Loosely speaking, since the ℓ_1 arc length accounts for the entire history of the path up until the current point, the (limiting) stagewise algorithm is less "willing" to produce wiggly estimates.

  Despite these differences, stagewise estimates tend to perform competitively with lasso estimates in terms of test error, and this is true even with highly correlated predictor variables, when the stagewise and lasso paths are very different (such statements are based on simulations, and not theory; see Hastie et al. (2007), Knudsen (2013)). This is a critical point, as it suggests that stagewise should be considered as an effective tool for regularized estimation, apart from any link to a convex problem. We return to this idea throughout the paper.

- *General convex loss functions.* Fortunately, the stagewise method extends naturally to sparse modeling in other settings, beyond Gaussian regression. Let $f : \mathbb{R}^p \to \mathbb{R}$ be a differentiable convex loss function, e.g., $f(\beta) = \frac{1}{2}\|y - X\beta\|_2^2$ for the regression setting. Beginning again with $\beta^{(0)} = 0$, the analogy of the stagewise steps in (1), (2) for the present general setting are

$$\beta^{(k)} = \beta^{(k-1)} - \epsilon \cdot \mathrm{sign}\big(\nabla_i f(\beta^{(k-1)})\big) \cdot e_i, \tag{7}$$

$$\text{where } i \in \operatorname*{argmax}_{j=1,\ldots p} |\nabla_j f(\beta^{(k-1)})|. \tag{8}$$

  That is, at each iteration we update $\beta^{(k)}$ in the direction opposite to the largest component of the gradient (largest in absolute value). Note that this reduces to the usual update rules (1),

(2) when $f(\beta) = \frac{1}{2}\|y - X\beta\|_2^2$. Rosset et al. (2004) studied the stagewise routine (7), (8), and its connection to the ℓ_1-constrained estimate

$$\hat{\beta}(t) = \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \; f(\beta) \;\; \text{subject to} \;\; \|\beta\|_1 \leq t. \tag{9}$$

Similar to the result for lasso regression, these authors prove that if the solution $\hat{\beta}(t)$ in (9) has monotone coordinate paths, then under mild conditions[3] on f, the stagewise paths given by (7), (8) converge to the path $\hat{\beta}(t)$ as $\epsilon \to 0$. This covers, e.g., the cases of logistic regression and Poisson regression losses, with predictor variables X in general position. The same general message, as in the linear regression setting, applies here: compared to the relatively complex optimization problem (9), the stagewise algorithm (7), (8) is very simple. The most (or really, the only) advanced part of each iteration is the computation of the gradient $\nabla f(\beta^{(k-1)})$; in the logistic or Poisson regression settings, the components of $\nabla f(\beta^{(k-1)})$ are given by

$$\nabla_j f(\beta^{(k-1)}) = X_j^T \big(y - \mu(\beta^{(k-1)})\big), \quad j = 1, \ldots p,$$

where $y \in \mathbb{R}^n$ is the outcome and $\mu(\beta^{(k-1)}) \in \mathbb{R}^n$ has components

$$\mu_i(\beta^{(k-1)}) = \begin{cases} 1/[1 + \exp(-(X\beta^{(k-1)})_i)] & \text{for logistic regression} \\ \exp((X\beta^{(k-1)})_i) & \text{for Poisson regression} \end{cases}, \quad i = 1, \ldots n.$$

Its precise connection to the ℓ_1-constrained optimization problem (9) for monotone paths is encouraging, but even outside of this case, the simple and efficient stagewise algorithm (7), (8) produces regularized estimates deserving of attention in their own right.

- *Forward-backward stagewise.* Zhao & Yu (2007) examined a novel modification of forward stagewise, under a general loss function f: at each iteration, their proposal takes a backward step (i.e., moves a component of $\beta^{(k)}$ towards zero) if this would decrease the loss function by a sufficient amount ξ; otherwise it takes a forward step as usual. The authors prove that, as long as the parameter ξ used for the backward steps scales as $\xi = o(\epsilon)$, the path from this forward-backward stagewise algorithm converges to the solution path in (9) as $\epsilon \to 0$. The important distinction here is that their result does not assume monotonicity of the coordinate paths in (9). (It does, however, assume that the loss function f is strongly convex—in the linear regression setting, $f(\beta) = \frac{1}{2}\|y - X\beta\|_2^2$, this is equivalent to assuming that $X \in \mathbb{R}^{n \times p}$ has linearly independent predictors, which requires $n \geq p$).[4] The forward-backward stagewise algorithm hence provides another way to view the connection between (the usual) forward stagewise steps (7), (8) and the ℓ_1-regularized optimization problem (9): the forward stagewise path is an approximation to the solution path in (9) given by skipping the requisite backward steps needed to correct for nonmonotonicities.

Clearly, there has been some fairly extensive work connecting the stagewise estimates (1), (2) and the lasso estimate (3), or more generally, the stagewise estimates (7), (8) and the ℓ_1-constrained estimate (9). Still, however, this connection seems mysterious. Both methods produce a regularization path, with a fully sparse model on one end, and a fully dense model on the other—but beyond this basic degree of similarity, why should we expect the stagewise path (7), (8) and the ℓ_1 regularization path (9) to be so closely related? The work referenced above gives a mathematical treatment of this question, and we feel, does not provide much intuition. In fact, there is a simple interpretation of the forward stagewise algorithm that explains its connection to the lasso problem, seen next.

[3] Essentially, Rosset et al. (2004) assume that conditions on f that imply a unique solution in (9), and allow for a second order Taylor expansion of f. Such conditions are that $f(\beta) = h(X\beta)$, with h twice differentiable and strictly convex, and X having columns in general position.

[4] It is also worth pointing out that the type of convergence considered by Zhao & Yu (2007) is stronger than that considered by Efron et al. (2004) and Rosset et al. (2004). The former authors prove that, under suitable conditions, the entire stagewise path converges globally to the lasso solution path; the latter authors only prove a local type of convergence, that has to do with the limiting stagewise and lasso directions at any fixed point along the path.

2.2 A new perspective on forward stagewise regression

We start by rewriting the steps (7), (8) for the stagewise algorithm, under a general loss f, as

$$\begin{gathered}\beta^{(k)} = \beta^{(k-1)} + \Delta, \\ \text{where } \Delta = -\epsilon \cdot \text{sign}\big(\nabla_i f(\beta^{(k-1)})\big) \cdot e_i, \\ \text{and } |\nabla_i f(\beta^{(k-1)})| = \|\nabla f(\beta^{(k-1)})\|_\infty.\end{gathered}$$

As $\nabla_i f(\beta^{(k-1)})$ is maximal in absolute value among all components of the gradient, the quantity $\text{sign}(\nabla_i f(\beta^{(k-1)})) \cdot e_i$ is a subgradient of the ℓ_∞ norm evaluated at $\nabla f(\beta^{(k-1)})$:

$$\Delta \in -\epsilon \cdot \Big(\partial \|x\|_\infty \Big|_{x = \nabla f(\beta^{(k-1)})}\Big).$$

Using the duality between the ℓ_∞ and ℓ_1 norms,

$$\Delta \in -\epsilon \cdot \Big(\operatorname*{argmax}_{z \in \mathbb{R}^p} \langle \nabla f(\beta^{(k-1)}), z \rangle \;\text{ subject to }\; \|z\|_1 \leq 1\Big),$$

or equivalently,

$$\Delta \in \operatorname*{argmin}_{z \in \mathbb{R}^p} \langle \nabla f(\beta^{(k-1)}), z \rangle \;\text{ subject to }\; \|z\|_1 \leq \epsilon.$$

(Above, as before, the element notation emphasizes that the maximizer or minimizer is not necessarily unique.) Hence the forward stagewise steps (7), (8) satisfy

$$\beta^{(k)} = \beta^{(k-1)} + \Delta, \tag{10}$$

$$\text{where } \Delta \in \operatorname*{argmin}_{z \in \mathbb{R}^p} \langle \nabla f(\beta^{(k-1)}), z \rangle \;\text{ subject to }\; \|z\|_1 \leq \epsilon. \tag{11}$$

Written in this form, the stagewise algorithm exhibits a natural connection to the ℓ_1-regularized optimization problem (9). At each iteration, forward stagewise moves in a direction that minimizes the inner product with the gradient of f, among all directions constrained to have a small ℓ_1 norm; therefore, the sequence of stagewise estimates balance (small) decreases in the loss function f with (small) increases in the ℓ_1 norm, just like the solution path in (9), as the regularization parameter t increases. This intuitive perspective aside, the representation (10), (11) for the forward stagewise estimates is important because it inspires an analogous approach for general convex regularization problems. This was already presented in Algorithm 2, and next we discuss it further.

2.3 Basic properties of the general stagewise procedure

Recall the general minimization problem in (4), where we assume that the loss function f is convex and differentiable, and the regularizer g is convex. It can now be seen that the steps (5), (6) in the general stagewise procedure in Algorithm 2 are directly motivated by the forward stagewise steps, as expressed in (10), (11). The explanation is similar to that given above: as we repeat the steps of the algorithm, the iterates are constructed to decrease the loss function f (by following its negative gradient) at the cost of a small increase in the regularizer g. In this sense, the stagewise algorithm navigates the trade-off between minimizing f and g, and produces an approximate regularization path for (4), i.e., the kth iterate $x^{(k)}$ approximately solves problem (4) with $t = g(x^{(k)})$.

From our work at the end of the last subsection, it is clear that forward stagewise regression (7), (8), or equivalently (10), (11), is a special case of the general stagewise procedure, applied to the ℓ_1-regularized problem (9). Moreover, the general stagewise procedure can be applied in many other settings, well beyond ℓ_1 regularization, as we show in the next section. Before presenting these applications, we now make several basic remarks.

- *Initialization and termination.* In many cases, initializing the algorithm is easy: if $g(x) = 0$ implies $x = 0$ (e.g., this is true when g is a norm), then we can start the stagewise procedure at $t_0 = 0$ and $x^{(0)} = 0$. In terms of a stopping criterion, a general strategy for (approximately) tracing a full solution path is to stop the algorithm when $g(x^{(k)})$ does not change very much between successive iterations. If instead the algorithm has been terminated upon reaching some maximum number of iterations or some maximum value of $g(x^{(k)})$, and more iterations are desired, then the algorithm can surely be restarted from the last reached iterate $x^{(k)}$.

- *First-order justification.* If g satisfies the triangle inequality (again, e.g., it would as a norm), then the increase in the value of g between successive iterates is bounded by ϵ:

$$g(x^{(k)}) \leq g(x^{(k-1)}) + g(\Delta) \leq g(x^{(k-1)}) + \epsilon.$$

  Furthermore, we can give a basic (and heuristic) justification of the stagewise steps (5), (6). Consider the minimization problem (4) at the parameter $t = g(x^{(k-1)}) + \epsilon$; we can write this as

$$\hat{x}(t) \in \operatorname*{argmin}_{x \in \mathbb{R}^n} f(x) - f(x^{(k-1)}) \;\text{ subject to }\; g(x) - g(x^{(k-1)}) \leq \epsilon,$$

  and then reparametrize as

$$\hat{x}(t) = x^{(k-1)} + \Delta^*, \tag{12}$$

$$\Delta^* \in \operatorname*{argmin}_{z \in \mathbb{R}^n} f(x^{(k-1)} + z) - f(x^{(k-1)}) \;\text{ subject to }\; g(x^{(k-1)} + z) - g(x^{(k-1)}) \leq \epsilon. \tag{13}$$

  We now modify the problem (13) in two ways: first, we replace the objective function in (13) with its first-order (linear) Taylor approximation around $x^{(k-1)}$,

$$\langle \nabla f(x^{(k-1)}), z \rangle \approx f(x^{(k-1)} + z) - f(x^{(k-1)}), \tag{14}$$

  and second, we shrink the constraint set in (13) to

$$\{z \in \mathbb{R}^n : g(z) \leq \epsilon\} \subseteq \{z \in \mathbb{R}^n : g(x^{(k-1)} + z) - g(x^{(k-1)}) \leq \epsilon\},$$

  since, as noted earlier, any element of the left-hand side above is an element of the right-hand side by the triangle inequality. These two modifications define a different update direction

$$\Delta \in \operatorname*{argmin}_{z \in \mathbb{R}^n} \langle \nabla f(x^{(k-1)}), z \rangle \;\text{ subject to }\; g(z) \leq \epsilon,$$

  which is exactly the direction (6) in the general stagewise procedure. Hence the stagewise algorithm chooses Δ as above, rather than choosing the actual direction Δ^* in (13), to perform an update step from $x^{(k-1)}$. This update results in a feasible point $x^{(k)} = x^{(k-1)} + \Delta$ for the problem (4) at $t = g^{(k-1)} + \epsilon$; of course, the point $x^{(k)}$ is not necessarily optimal, but as ϵ gets smaller, the first-order Taylor approximation in (14) becomes tighter, so one would imagine that the point $x^{(k)}$ becomes closer to optimal.

- *Dual update form.* If g is a norm, then the update direction defined in (6) can be expressed more succinctly in terms of the dual norm $g^*(x) = \max_{g(z) \leq 1} x^T z$. We write

$$\begin{aligned} \Delta &\in -\epsilon \cdot \Big(\operatorname*{argmax}_{z \in \mathbb{R}^n} \langle \nabla f(x^{(k-1)}), z \rangle \;\text{ subject to }\; g(z) \leq 1 \Big) \\ &= -\epsilon \cdot \partial g^* \big(\nabla f(x^{(k-1)}) \big), \end{aligned} \tag{15}$$

  i.e., the direction Δ is $-\epsilon$ times a subgradient of the dual norm g^* evaluated at $\nabla f(x^{(k-1)})$. This is a useful observation, since many norms admit a known dual norm with known subgradients; we will see examples of this in the coming section.

- *Invariance around ∇f.* The level of difficulty associated with computing the update direction, i.e., in solving problem (6), depends entirely on g and not on f at all (assuming that ∇f can be readily computed). We can think of Δ as an operator on $\mathbb{R}^n$:

$$\Delta(x) \in \operatorname*{argmin}_{z \in \mathbb{R}^n} \langle x, z \rangle \;\; \text{subject to} \;\; g(z) \leq \epsilon. \tag{16}$$

  This operator $\Delta(\cdot)$ is often called the *linear minimization oracle* associated with the function g, in the optimization literature. At each input x, it returns a minimizer of the problem in (16). Provided that $\Delta(\cdot)$ can be expressed in closed-form—which is fortuitously the case for many common statistical optimization problems, as we will see in the sections that follow—the stagewise update step (5) simply evaluates this operator at $\nabla f(x^{(k-1)})$, and adds the result to $x^{(k-1)}$:

$$x^{(k)} = x^{(k-1)} + \Delta\big(\nabla f(x^{(k-1)})\big).$$

  An analogy can be drawn here to the proximal operator in proximal gradient descent, used for minimizing the composite function $f + g$, where f is smooth but g is (possibly) nonsmooth. The proximal operator is defined entirely in terms of g, and as long as it can be expressed analytically, the generalized gradient update for $x^{(k)}$ simply uses the output of this operator at $\nabla f(x^{(k-1)})$.

- *Unbounded stagewise steps.* Suppose that g is a seminorm, i.e., it satisfies $g(ax) = |a|g(x)$ for $a \in \mathbb{R}$, and $g(x + y) \leq g(x) + g(y)$, but g can have a nontrivial null space, $N_g = \{x \in \mathbb{R}^n : g(x) = 0\}$. In this case, the stagewise update step in (5) can be unbounded; in particular, if

$$\langle \nabla f(x^{(k)}), z \rangle \neq 0 \quad \text{for some } z \in N_g, \tag{17}$$

  then we can drive $\langle \nabla f(x^{(k)}), z \rangle \to -\infty$ along a sequence with $g(z) = 0$, and so the stagewise update step would be clearly undefined. Fortunately, a simple modification of the general stagewise algorithm can account for this problem. Since we are assuming that g is a seminorm, the set N_g is a linear subspace. To initialize the general stagewise algorithm at say $t_0 = 0$, therefore, we solve the linearly constrained optimization problem

$$x^{(0)} \in \operatorname*{argmin}_{x \in N_g} f(x).$$

  In subsequent stagewise steps, we then restrict the updates to lie in the subspace orthogonal to N_g. That is, to be explicit, we replace (5) (6) in Algorithm 2 with

$$x^{(k)} = x^{(k-1)} + \Delta, \tag{18}$$

$$\text{where } \Delta \in \operatorname*{argmin}_{z \in N_g^\perp} \langle \nabla f(x^{(k-1)}), z \rangle \;\; \text{subject to} \;\; g(z) \leq \epsilon, \tag{19}$$

  where $N_g^\perp$ denotes the orthocomplement of N_g. We will see this modification, e.g., put to use for the quadratic regularizer $g(\beta) = \beta^T Q \beta$, where Q is positive semidefinite and singular.

Some readers may wonder why we are working with the constrained problem (4), and not

$$\hat{x}(\lambda) \in \operatorname*{argmin}_{x \in \mathbb{R}^n} f(x) + \lambda g(x), \tag{20}$$

where $\lambda \geq 0$ is now the regularization parameter, and is called the Lagrange multiplier associated with g. It is probably more common in the current statistics and machine learning literature for optimization problems to be expressed in the Lagrange form (20), rather than the constrained form (4). The solution paths of (4) and (20) (given by varying t and λ in their respective problems) are

not necessarily equal for general convex functions f and g; however, they are equal under very mild assumptions[5], which hold for all of the examples visited in this paper. Therefore, there is not an important difference in terms of studying (4) versus (20). We choose to focus on (4) as we feel that the intuition for stagewise algorithms is easier to see with this formulation.

2.4 Related work

There is a lot of work related to the proposal of this paper. Readers familiar with optimization will likely identify the general stagewise procedure, in Algorithm 2, as a particular type of (normalized) *steepest descent*. Steepest descent is an iterative algorithm for minimizing a smooth convex function f, in which we update the current iterate in a direction that minimizes the inner product with the gradient of f (evaluated at the current iterate), among all vectors constrained to have norm $\|\cdot\|$ bounded by 1 (e.g., see Boyd & Vandenberghe (2004)); the step size for the update can be chosen in any one of the usual ways for descent methods. Note that gradient descent is simply a special case of steepest descent with $\|\cdot\| = \|\cdot\|_2$ (modulo normalizing factors). Meanwhile, the general stagewise algorithm is just steepest descent with $\|\cdot\| = g(\cdot)$, and a constant step size ϵ. It is important to point out that our interest in the general stagewise procedure is different from typical interest in steepest descent. In the classic usage of steepest descent, we seek to minimize a differentiable convex function f; our choice of norm $\|\cdot\|$ affects the speed with which we can find such a minimizer, but under weak conditions, any choice of norm will eventually bring us to a minimizer nonetheless. In the general stagewise algorithm, we are not really interested in the final minimizer itself, but rather, the path traversed in order to get to this minimizer. The stagewise path is composed of iterates that have interesting statistical properties, given by gradually balancing f and g; choosing different functions g will lead to generically different paths. Focusing on the path, instead of its endpoint, may seem strange to a researcher in optimization, but it is quite natural for researchers in statistics and machine learning.

Another method related to our general stagewise proposal is the *Frank-Wolfe algorithm* (Frank & Wolfe 1956), used to minimize a differentiable convex function f over a convex set C. Similar to (projected) gradient descent, which iteratively minimizes local quadratic approximations of f over C, the Frank-Wolfe algorithm iteratively minimizes local linear approximations of f over C. In a recent paper, Jaggi (2013) shed light on Frank-Wolfe as an efficient, scalable algorithm for modern machine learning problems. For a single value of the regularization parameter t, the Frank-Wolfe algorithm can be used to solve problem (4), taking as the constraint set $C = \{x : g(x) \leq t\}$; the Frank-Wolfe steps here look very similar to the general stagewise steps (5), (6), but an important distinction is that the iterates from Frank-Wolfe result in a single estimate, rather than each iterate constituting its own estimate along the regularization path, as in the general stagewise procedure. This connection deserves more discussion, and so we dedicate a subsection of the appendix to it: see Appendix A.1. Other well-known methods based on local linearization are *cutting-plane* (Kelley 1960) and *bundle* (Hiriart-Urruty & Lemarechal 1993) methods. Teo et al. (2007) present a general bundle method for regularized risk minimization that is particularly relevant to our proposal (see also Teo et al. (2010)); this is similar to the Frank-Wolfe approach in that it solves the problem (4) at a fixed value of the parameter t (one difference is that its local linearization steps are based on the entire history of previous iterates, instead of just the single last iterate). For brevity, we do not conduct a detailed comparison between their bundle method and our general stagewise procedure, though we believe it would be interesting to do so.

Yet another class of methods that are highly relevant to our proposal are *boosting* procedures. Boosting algorithms are iterative in form, and we typically think of them as tracing out a sequence of estimates, just like our general stagewise algorithm (and unlike the iterative algorithms described above, e.g., steepest descent and Frank-Wolfe, which we tend to think of as culminating in a single

[5]For example, it is enough to assume that $g \geq 0$, and that for all parameters $t, \lambda \geq 0$, the solution sets of (4), (20) are nonempty.

estimate). The literature on boosting is vast; see, e.g., Hastie et al. (2009) or Buhlmann & Yu (2010) for a nice review. Among boosting methods, *gradient boosting* (Friedman 2001) most closely parallels forward stagewise fitting. Consider a setup in which our weak learners are the individual predictor variables $X_1, \dots X_p$, and the loss function is $L(X\beta) = f(\beta)$. The gradient boosting updates, using a shrinkage factor ϵ, are most commonly expressed in terms of the fitted values, as in

$$X\beta^{(k)} = X\beta^{(k-1)} + \epsilon \cdot \alpha_i X_i, \tag{21}$$

$$\text{where } \alpha_i \in \operatorname*{argmin}_{\alpha \in \mathbb{R}} L(X\beta^{(k-1)} + \alpha X_i), \tag{22}$$

$$\text{and } i \in \operatorname*{argmin}_{j=1,\dots p} \left(\min_{\alpha \in \mathbb{R}} \| - \nabla L(X\beta^{(k-1)}) - \alpha X_j \|_2^2 \right). \tag{23}$$

The step (23) selects the weak learner X_i that best matches the negative gradient, $-\nabla L(X\beta^{(k-1)})$, in a least squares sense; the step (22) chooses the coefficient α_i of X_i via line search. If we assume that the variables have been scaled to have unit norm, $\|X_j\|_2 = 1$ for $j = 1, \dots p$, then it is easy to see that (23) is equivalent to

$$i \in \operatorname*{argmax}_{j=1,\dots p} |X_j^T \nabla L(X\beta^{(k-1)})| = \operatorname*{argmax}_{j=1,\dots p} |\nabla_j f(\beta^{(k-1)})|,$$

which is exactly the same selection criterion used by forward stagewise under the loss function f, as expressed in (8). Therefore, at a given iteration, gradient boosting and forward stagewise choose the next variable i in the same manner, and only differ in their choice of the coefficient of X_i in the constructed additive model. The gradient boosting update in (21) adds $\epsilon \cdot \alpha_i X_i$ to the current model, where α_i is chosen by line search in (22); meanwhile, the forward stagewise update in (7) can be expressed as

$$X\beta^{(k)} = X\beta^{(k-1)} + \epsilon \cdot s_i X_i, \tag{24}$$

where $s_i = -\text{sign}(\nabla_i f(\beta^{(k-1)})$, a simple choice of coefficient compared to α_i. Because α_i is chosen by minimizing the loss function along the direction defined by X_i (anchored at $X\beta^{(k-1)}$), gradient boosting is even more greedy than forward stagewise, but practically there is not a big difference between the two, especially when ϵ is small. In fact, the distinction between (21) and (24) is slight enough that several authors refer to forward stagewise as a boosting procedure, e.g., Rosset et al. (2004), Zhao & Yu (2007), and Buhlmann & Yu (2010) refer to forward stagewise as *ϵ-boosting*.

The tie between boosting and forward stagewise suggests that we might be able to look at our general stagewise proposal through the lens of boosting, as well. Above we compared boosting and forward stagewise for the problem of sparse estimation; in this problem, deciding on the universe of weak learners for gradient boosting is more or less straightforward, as we can use the variables $X_1, \dots X_p$ themselves (or, e.g., smooth marginal transformations of these variables for sparse non-parametric estimation). This works because each iteration of gradient boosting adds a single weak learner to the fitted model, so the model is sparse in the early stages of the algorithm, and becomes increasingly dense as the algorithm proceeds. However, for more complex problems (beyond sparse estimation), specifying a universe of weak learners is not as straightforward. Consider, e.g., matrix completion or image denoising—what kind of weak learners would be appropriate here? At a broad level, our general stagewise procedure offers a prescription for a class of weak learners based on the regularizer g, through the definition of Δ in (6). Such weak learners seem intuitively reasonable in various problem settings: they end up being groups of variables for group-structured estimation problems (see Section 3.1), rank 1 matrices for matrix completion (Section 3.3), and pixel contrasts for image denoising (Section 3.5). This may lead to an interesting perspective on gradient boosting with an arbitrary regularization scheme, though we do not explore it further.

Finally, the form of the update Δ in (6) sets our work apart from other general path tracing procedures. Zhao & Yu (2007) and Friedman (2008) propose approximate path following methods for optimization problems whose regularizers extend beyond the ℓ_1 norm, but their algorithms only

update one component of the estimate at a time (which corresponds to utilizing individual variables as weak learners, in the boosting perspective); on the other hand, our general stagewise procedure specifically adapts its updates to the regularizer of concern g. We note that, in certain special cases (i.e., for certain regularizers g), our proposed algorithm bears similarities to existing algorithms in the literature: for ridge regularization, our proposal is similar to gradient-directed path following, as studied in Friedman & Popescu (2004) and Ramsay (2005), and for ℓ_1/ℓ_2 multitask learning, our stagewise algorithm is similar to the block-wise path following method of Obozinski et al. (2010).

3 Applications of the general stagewise framework

3.1 Group-structured regularization

We begin by considering the group-structured regularization problem

$$\hat{\beta}(t) \in \operatorname*{argmin}_{\beta \in \mathbb{R}^p} f(\beta) \;\text{ subject to }\; \sum_{j=1}^{G} w_j \|\beta_{\mathcal{I}_j}\|_2 \leq t, \tag{25}$$

where the index set $\{1, \ldots p\}$ has been partitioned into G groups $\mathcal{I}_1, \ldots \mathcal{I}_G$, $\beta_{\mathcal{I}_j} \in \mathbb{R}^{p_j}$ denotes the components of $\beta \in \mathbb{R}^p$ for the jth group, and $w_1, \ldots w_G \geq 0$ are fixed weights. The loss f is kept as a generic differentiable convex function—this is because, as explained in Section 2.3, the stagewise updates are invariant around ∇f, in terms of their computational form.

Note that the *group lasso* problem (Bakin 1999, Yuan & Lin 2006) is a special case of (25). In the typical group lasso regression setup, we observe an outcome $y \in \mathbb{R}^n$ and predictors $X \in \mathbb{R}^{n \times p}$, and the predictor variables admit some natural grouping $\mathcal{I}_1, \ldots \mathcal{I}_G$. To perform group-wise variable selection, one can use the group lasso estimator, defined as in (25) with

$$f(\beta) = \frac{1}{2}\Big\|y - \sum_{j=1}^{G} X_{\mathcal{I}_j}\beta_{\mathcal{I}_j}\Big\|_2^2 \quad \text{and} \quad w_j = \sqrt{p_j},\;\; j = 1, \ldots G,$$

where $X_{\mathcal{I}_j} \in \mathbb{R}^{n \times p_j}$ is the predictor matrix for group j, and $p_j = |\mathcal{I}_j|$ is the size of the group j. The same idea clearly applies outside of the linear regression setting (e.g., see Meier et al. (2008) for a study of the group lasso regularization in logistic regression).

A related yet distinct problem is that of *multitask learning*. In this setting we consider not one but multiple learning problems, or tasks, and we want to select a common set of variables that are important across all tasks. A popular estimator for this purpose is based on ℓ_1/ℓ_2 regularization (Argyriou et al. 2006, Obozinski et al. 2010), and also fits into the framework (25): the loss function f becomes the sum of the losses across the tasks, and the groups $\mathcal{I}_1, \ldots \mathcal{I}_G$ collect the coefficients corresponding to the same variables across tasks. For example, in multitask linear regression, we write $y^{(i)} \in \mathbb{R}^n$ for the outcome, $X^{(i)} \in \mathbb{R}^{n \times m}$ for the predictors, and $\beta^{(i)}$ the coefficients for the ith task, $i = 1, \ldots r$. We form a global coefficient vector $\beta = (\beta^{(1)}, \ldots \beta^{(m)}) \in \mathbb{R}^p$, where $p = m \cdot r$, and form groups $\mathcal{I}_1, \ldots \mathcal{I}_m$, where $\mathcal{I}_j$ collects the coefficients of predictor variable j across the tasks. The ℓ_1/ℓ_2 regularized multitask learning estimator is then defined as in (25) with

$$f(\beta) = \frac{1}{2}\sum_{i=1}^{r} \|y^{(i)} - X^{(i)}\beta^{(i)}\|_2^2 \quad \text{and} \quad w_j = 1,\;\; j = 1, \ldots m,$$

where the default is to set all of the weights to 1, in the lack of any prior information about variable importance (note that the groups $\mathcal{I}_1, \ldots \mathcal{I}_m$ are all the same size here).

The general stagewise algorithm, Algorithm 2, does not make any distinction between cases such as the group lasso and multitask learning problems; it only requires f to be a convex and smooth function. To initialize the algorithm for the group regularized problem (25), we can take $t_0 = 0$ and $\beta^{(0)} = 0$. The next lemma shows how to calculate the appropriate update direction Δ in (6).

Lemma 1. *For $g(\beta) = \sum_{j=1}^G w_j \|\beta_{\mathcal{I}_j}\|_2$, the general stagewise procedure in Algorithm 2 repeats the updates $\beta^{(k)} = \beta^{(k-1)} + \Delta$, where Δ can be computed as follows: first find i such that*

$$\frac{\|(\nabla f)_{\mathcal{I}_i}\|_2}{w_i} = \max_{j=1,\ldots G} \frac{\|(\nabla f)_{\mathcal{I}_j}\|_2}{w_j}, \tag{26}$$

where we abbreviate $\nabla f = \nabla f(\beta^{(k-1)})$, then let

$$\Delta_{\mathcal{I}_j} = 0 \quad \textit{for all } j \neq i, \tag{27}$$

$$\Delta_{\mathcal{I}_i} = \frac{-\epsilon \cdot (\nabla f)_{\mathcal{I}_i}}{w_i \|(\nabla f)_{\mathcal{I}_i}\|_2}. \tag{28}$$

We omit the proof; it follows straight from the KKT conditions for (6), with g as defined in the lemma. Computation of Δ in (26), (27), (28) is very cheap, and requires $O(p)$ operations. To rephrase: at the kth iteration, we simply find the group i such that the corresponding block of the gradient $\nabla f(\beta^{(k-1)})$ has the largest ℓ_2 norm (after scaling appropriately by the weights). We then move the coefficients for group i in a direction opposite to this gradient value; for all other groups, we leave their coefficients untouched (note that, if a group has not been visited by past update steps, then this means leaving its coefficients identically equal to zero). The outputs of the stagewise algorithm therefore match our intuition about the role of the constraint in (25)—for some select groups, all coefficients are set to nonzero values, and for other groups, all coefficients are set to zero. That the actual solution in (25) satisfies this intuitive property can be verified by examining its own KKT conditions.

Looking back at Figure 2, the first row compares the exact solution and stagewise paths for a group lasso regression problem. The stagewise path was computed using 300 steps with $\epsilon = 0.01$, and shows strong similarities to the exact group lasso path. In other problem instances, say, when the predictors across different groups are highly correlated, the group lasso coefficient paths can behave wildly with t, and yet the stagewise paths can appear much less wild and more stable. Later, in Section 4, we consider larger examples and give more thorough empirical comparisons.

3.2 Group-structured regularization with arbitrary norms

Several authors have considered group-based regularization using the ℓ_∞ norm in place of the usual ℓ_2 norm (e.g., see Turlach et al. (2005) for such an approach in multitask learning). To accomodate this and other general group-structured regularization approaches, we consider the problem

$$\hat{\beta}(t) \in \operatorname*{argmin}_{\beta \in \mathbb{R}^p} f(\beta) \;\text{ subject to }\; \sum_{j=1}^G w_j h_j(\beta_{\mathcal{I}_j}) \leq t, \tag{29}$$

where each h_j is an arbitrary norm. Let h_j^* denote the dual norm of h_j; e.g., if $h_j(x) = \|x\|_{q_j}$, then $h_j^*(x) = \|x\|_{r_j}$, where $1/q_j + 1/r_j = 1$. Similar to the result in Lemma 1, the stagewise updates for problem (29) take a simple group-based form.

Lemma 2. *For $g(\beta) = \sum_{j=1}^G w_j h_j(\beta_{\mathcal{I}_j})$, the general stagewise procedure in Algorithm 2 repeats the updates $\beta^{(k)} = \beta^{(k-1)} + \Delta$, where Δ can be computed as follows: first find i such that*

$$\frac{h_i^*\big((\nabla f)_{\mathcal{I}_i}\big)}{w_i} = \max_{j=1,\ldots G} \frac{h_j^*\big((\nabla f)_{\mathcal{I}_j}\big)}{w_j},$$

where we abbreviate $\nabla f = \nabla f(\beta^{(k-1)})$, then let

$$\Delta_{\mathcal{I}_j} = 0 \quad \textit{for all } j \neq i,$$

$$\Delta_{\mathcal{I}_i} \in -\frac{\epsilon}{w_i} \cdot \partial h_i^*\big((\nabla f)_{\mathcal{I}_i}\big).$$

Again we omit the proof; it follows from the KKT conditions for (6). Indeed, Lemma 2 covers Lemma 1 as a special case, recalling that the ℓ_2 norm is self-dual. Also, recalling that the ℓ_∞ and ℓ_1 norms are dual, Lemma 2 says that the stagewise algorithm for $g(\beta) = \sum_{j=1}^G w_j \|\beta_{\mathcal{I}_j}\|_\infty$ first finds i such that

$$\frac{\|(\nabla f)_{\mathcal{I}_i}\|_1}{w_i} = \max_{j=1,\ldots G} \frac{\|(\nabla f)_{\mathcal{I}_j}\|_1}{w_j},$$

and then defines the update direction Δ by

$$\begin{aligned} \Delta_{\mathcal{I}_j} &= 0 \quad \text{for all } j \neq i, \\ \Delta_\ell &= -\frac{\epsilon}{w_i} \cdot \begin{cases} 0 & \text{for } \ell \in \mathcal{I}_i,\ (\nabla f)_\ell = 0 \\ \text{sign}\big((\nabla f)_\ell\big) & \text{for } \ell \in \mathcal{I}_i,\ (\nabla f)_\ell \neq 0. \end{cases} \end{aligned}$$

More broadly, Lemma 2 provides a general prescription for deriving the stagewise updates for regularizers that are block-wise sums of norms, as long as we can compute subgradients of the dual norms. For example, the norms in consideration could be a mix of ℓ_p norms, matrix norms, etc.

3.3 Trace norm regularization

Consider a class of optimization problems over matrices,

$$\hat{B}(t) \in \operatorname*{argmin}_{B \in \mathbb{R}^{m \times n}} f(B) \ \text{ subject to } \ \|B\|_* \leq t, \tag{30}$$

where $\|B\|_*$ denotes the trace norm (also called the nuclear norm) of a matrix B, i.e., the sum of its singular values. Perhaps the most well-known example of trace norm regularization comes from the problem of *matrix completion* (e.g., see Candes & Recht (2009), Candes & Tao (2010), Mazumder et al. (2010)). Here the setup is that we only partially observe entries of a matrix $Y \in \mathbb{R}^{m \times n}$—say, we observe all entries $(i, j) \in \Omega$—and we seek to estimate the missing entries. A natural estimator for this purpose (studied by, e.g., Mazumder et al. (2010)) is defined as in (30) with

$$f(B) = \frac{1}{2} \sum_{(i,j) \in \Omega} (Y_{ij} - B_{ij})^2.$$

The trace norm also appears in interesting examples beyond matrix completion. For example, Chen & Ye (2014) consider regularization with the trace norm in multiple nonparametric regression, and Harchaoui et al. (2012) consider it in large-scale image classification.

The general stagewise algorithm applied to the trace norm regularization problem (30) can be initialized with $t_0 = 0$ and $B^{(0)} = 0$, and the update direction in (6) is now simple and efficient.

Lemma 3. *For $g(B) = \|B\|_*$, the general stagewise procedure in Algorithm 2 repeats the updates $\beta^{(k)} = \beta^{(k-1)} + \Delta$, where*

$$\Delta = -\epsilon \cdot uv^T, \tag{31}$$

with u, v being leading left and right singular vectors, respectively, of $\nabla f(B^{(k-1)})$.

The proof relies on the fact that the dual of the trace norm $g(B) = \|B\|_*$ is the spectral norm $g^*(B) = \|B\|_2$, and then invokes the representation (15) for stagewise estimates. For the stagewise update direction (31), we need to compute the leading left and right singular vectors u, v of the $m \times n$ matrix $\nabla f(B^{(k-1)})$—these are the left and right singular vectors corresponding to the top singular value of $\nabla f(B^{(k-1)})$. Assuming that $\nabla f(B^{(k-1)})$ has a distinct largest singular value, this can be done, e.g., using the power method: letting $A = \nabla f(B^{(k-1)})$, we first run the power method on the $m \times m$ matrix AA^T, or the $n \times n$ matrix $A^T A$, depending on whichever is smaller. This gives us either u or v; to recover the other, we then simply use matrix multiplication: $v = A^T u / \|A^T u\|_2$

or $u = Av/\|Av\|_2$. The power method is especially efficient if $A = \nabla f(B^{(k-1)})$ is sparse (each iteration being faster), or has a large spectral gap (fewer iterations required until convergence). Of course, alternatives to the power method can be used for computing the leading singular vectors of $\nabla f(B^{(k-1)})$, such as methods based on inverse iterations, Rayleigh quotients, or QR iterations; see, e.g., Golub & Van Loan (1996).

In the second row of Figure 2, the exact and stagewise paths for are shown matrix completion problem, where the stagewise paths were computed using 500 steps with $\epsilon = 0.05$. While the two sets of paths appear fairly similar, we note that it is harder to judge the degree of similarity between the two in the matrix completion context. Here, the coordinate paths correspond to entries in the estimated matrix $\hat{B}$, and their roles are not as clear as they are in, say, in a regression setting, where the coordinate paths correspond to the coefficients of individual variables. In other words, it is difficult to interpret the slight differences between the exact and stagewise paths in the second row of Figure 2, which present themselves as the trace norm grows large. Therefore, to get a sense for the effect of these differences, we might compare the mean squared error curves generated by the exact and stagewise estimates. This is done in depth in Section 4.

3.4 Quadratic regularization

Consider problems of the form

$$\hat{\beta}(t) \in \operatorname*{argmin}_{\beta \in \mathbb{R}^p} f(\beta) \;\text{ subject to }\; \beta^T Q \beta \leq t, \tag{32}$$

where $Q \succeq 0$, a positive semidefinite matrix. The quadratic regularizer in (32) encompasses several common statistical tasks. When $Q = I$, the regularization term $\beta^T\beta = \|\beta\|_2^2$ is well-known as *ridge* (Hoerl & Kennard 1970), or *Tikhonov* regularization (Tikhonov 1943). This regularizer shrinks the components of the solution $\hat{\beta}$ towards zero. In a (generalized) linear model setting with many predictor variables, such shrinkage helps control the variance of the estimated coefficients. Beyond this simple ridge case, roughness regularization in nonparametric regression often fits into the form (32), with Q not just the identity. For example, *smoothing splines* (Wahba 1990, Green & Silverman 1994) and *P-splines* (Eilers & Marx 1996) can both be expressed as in (32). To see this, suppose that $y_1, \ldots y_n \in \mathbb{R}$ are observed across input points $x_1, \ldots x_n \in \mathbb{R}$, and let $b_1, \ldots b_p$ denote the B-spline basis (of, say, cubic order) with knots at locations $z_1, \ldots z_p \in \mathbb{R}$. Smoothing splines use the inputs as knots, $z_1 = x_1, \ldots z_p = x_n$ (so that $p = n$); P-splines typically employ a (much) smaller number of knots across the range of $x_1, \ldots x_n \in \mathbb{R}$. Both estimators solve problem (32), with a loss function $f(\beta) = \frac{1}{2}\|y - B\beta\|_2^2$, and $B \in \mathbb{R}^{n \times p}$ having entries $B_{ij} = b_j(x_i)$, but the two use a different definition for Q: its entries are given by $Q_{ij} = \int b_i''(x) b_j''(x)\, dx$ in the case of smoothing splines, while $Q = D^T D$ in the case of P-splines, where D is the discrete difference operator of a given (fixed) integral order. Both estimators can be extended to the logistic or Poisson regression settings, just by setting f to be the logistic or Poisson loss, with natural parameter $\eta = B\beta$ (Green & Silverman 1994, Eilers & Marx 1996).

When Q is positive definite, the general stagewise algorithm, applied to (32), can be initialized with $t_0 = 0$ and $\beta^{(0)} = 0$. The update direction Δ in (6) is described by the following lemma.

Lemma 4. *For $g(\beta) = \beta^T Q \beta$, with Q a positive definite matrix, the general stagewise procedure in Algorithm 2 repeats the updates $\beta^{(k)} = \beta^{(k-1)} + \Delta$, where*

$$\Delta = -\sqrt{\epsilon} \cdot \frac{Q^{-1}\nabla f}{\sqrt{(\nabla f)^T Q^{-1} \nabla f}}, \tag{33}$$

and ∇f is an abbreviation for $\nabla f(\beta^{(k-1)})$.

The proof follows by checking the KKT conditions for (6). When $Q = I$, the update step (33) of the general stagewise procedure for quadratic regularization is computationally trivial, reducing to

$$\Delta = -\sqrt{\epsilon} \cdot \frac{\nabla f}{\|\nabla f\|_2}.$$

This yields fast, simple updates for ridge regularized estimators. For a general matrix Q, computing the update direction in (33) boils down to solving the linear equation

$$Qv = \nabla f(\beta^{(k-1)}) \tag{34}$$

in v. This is expensive for an arbitrary, dense Q; a single solve of the linear system (34) generally requires $O(p^3)$ operations. Of course, since the systems across all iterations involve the same linear operator Q, we could initially compute a Cholesky decomposition of Q (or a related factorization), requiring $O(p^3)$ operations, and then use this factorization to solve (34) at each iteration, requiring only $O(p^2)$ operations. While certainly more efficient than the naive strategy of separately solving each instance of (34), this is still not entirely desirable for large problems.

On the other hand, for several cases in which Q is structured or sparse, the linear system (34) can be solved efficiently. For example, if Q is banded with bandwidth d, then we can solve (34) in $O(pd^2)$ operations (actually, an initial Cholesky decomposition takes $O(pd^2)$ operations, and each successive solve with this decomposition then takes $O(pd)$ operations).

Importantly, the matrix Q is banded in both the smoothing spline and P-spline regularization cases: for smoothing splines, Q is banded because the B-spline basis functions have local support; for P-splines, Q is banded because the discrete difference operator is. However, some care must be taken in applying the stagewise updates in these cases, as Q is singular, i.e., positive semidefinite but not strictly positive definite. The stagewise algorithm needs to be modified, albeit only slightly, to deal with this issue—this modification was discussed in (18), (19) in Section 2.3, and here we summarize the implications for problem (32). First we compute the initial iterate to lie in null(Q), the null space of Q,

$$\beta^{(0)} \in \operatorname*{argmin}_{\beta \in \mathrm{null}(Q)} f(\beta). \tag{35}$$

For, e.g., P-splines with $Q = D^T D$, and D the discrete difference operator of order k, the space null(Q) is k-dimensional and contains (the evaluations of) all polynomial functions of order $k-1$. The stagewise algorithm is then initialized at such a point $\beta^{(0)}$ in (35), and $t_0 = 0$. For future iterations, note that when $\nabla f(\beta^{(k)})$ has a nontrivial projection onto null(Q), the stagewise update in (6) is undefined, since $\langle \nabla f(\beta^{(k)}), z \rangle$ can be made arbitrarily small along a direction z such that $z^T Q z = 0$. Therefore, we must further constrain the stagewise update to lie in the orthocomplement $\mathrm{null}(Q)^\perp = \mathrm{row}(Q)$, the row space of Q, as in

$$\Delta \in \operatorname*{argmin}_{z \in \mathrm{row}(Q)} \langle \nabla f(\beta^{(k-1)}), z \rangle \;\text{ subject to }\; z^T Q z \leq \epsilon.$$

It is not hard to check that, instead of (33), the update now becomes

$$\Delta = -\sqrt{\epsilon} \cdot \frac{Q^+ \nabla f}{\sqrt{(\nabla f)^T Q^+ \nabla f}}, \tag{36}$$

with Q^+ denoting the (Moore-Penrose) generalized inverse of Q.

From a computational perspective, the stagewise update in (36) for the rank deficient case does not represent more much work than that in (33) for the full rank case. With P-splines, e.g., we have $Q = D^T D$ where $D \in \mathbb{R}^{(n-k) \times n}$ is a banded matrix of full row rank. A short calculation shows that in this case

$$(D^T D)^+ = D^T (D D^T)^{-2} D,$$

i.e., applying Q^+ is computationally equivalent to two banded linear system solves and two banded matrix multiplications. Hence one stagewise update for P-spline regularization problems takes $O(p)$ operations (the bandwidth of D is a constant, $d = k + 1$), excluding computation of the gradient.

The third row of Figure 2 shows an example of logistic regression with ridge regularization, and displays the grossly similar exact solution and stagewise paths. Notably, the stagewise path here was constructed using only *15 steps*, with an effective step size $\sqrt{\epsilon} = 0.1$. This is a surprisingly small number of steps, especially compared to the numbers needed by stagewise in the examples (both small and large) from other regularization settings covered in this paper. As far as we can tell, this rough scaling appears to hold for ridge regularization problems in general—for such problems, the stagewise algorithm can be run with relatively large step sizes for small numbers of steps, and it will still produce statistically appealing paths. Unfortunately, this trend does not persist uniformly across all quadratic regularization problems; it seems that the ridge case ($Q = I$) is really a special one.

For a second example, we consider P-spline regularization, using both continuous and binomial outcomes. The left panel of Figure 3 displays an array of stagewise estimates, computed under P-spline regularization and a Gaussian regression loss. We generated $n = 100$ noisy observations $y_1, \ldots y_{100}$ from an underlying sinusoidal curve, sampled at input locations $x_1, \ldots x_{100}$ drawn uniformly over $[0, 1]$. The P-splines were defined using 30 equally spaced knots across $[0, 1]$, and the stagewise algorithm was run for 300 steps with $\sqrt{\epsilon} = 0.005$. The figure shows the spline approximations delivered by the stagewise estimates (from every 15th step along the path, for visibility) and the true sinusoidal curve overlayed as a thick dotted black line. We note that in this particular setting, the stagewise algorithm is not so interesting computationally, because each update step solves a banded linear system, and yet the exact solution can itself be computed at the same cost, at any regularization parameter value. The example is instead meant to portray that the stagewise algorithm can produce smooth and visually reasonable estimates of the underlying curve.

The right panel of Figure 3 displays an analogous example using $n = 100$ binary observations, $y_1, \ldots y_{100}$, generated according to the probabilities $p_i^* = 1/(1 + e^{-\mu(x_i)})$, $i = 1, \ldots 100$, where the inputs $x_1, \ldots x_{100}$ were sampled uniformly from $[0, 1]$, and μ is a smooth function. The probability curve $p^*(x) = 1/(1 + e^{-\mu(x)})$ is drawn as a thick dotted black line. We ran the stagewise algorithm under a logistic loss, with $\sqrt{\epsilon} = 0.005$, and for 300 steps; the figure plots the probability curves associated with the stagewise estimates (from every 15th step along the path, for visibility). Again, we can see that the fitted curves are smooth and visually reasonable. Computationally, the difficulty of the stagewise algorithm in this logistic setting is essentially the same as that in the previous Gaussian setting; all that changes is the computation of the gradient, which is an easy task. The exact solution, however, is more difficult to compute in this setting than the previous, and requires the use of iterative algorithm like Newton's method. This kind of computational invariance around the loss function, recall, is an advantage of the stagewise framework.

3.5 Generalized lasso regularization

In this last application, we study generalized ℓ_1 regularization problems,

$$\hat{\beta}(t) \in \operatorname*{argmin}_{\beta \in \mathbb{R}^p} f(\beta) \;\text{ subject to }\; \|D\beta\|_1 \leq t, \tag{37}$$

where D is a given matrix (it need not be square). The regularization term above is also called *generalized lasso* regularization, since it includes lasso regularization as a special case, with $D = I$, but also covers a number of other regularization forms (Tibshirani & Taylor 2011). For example, *fused lasso* regularization is encompassed by (37), with D chosen to be the edge incidence matrix of some graph G, having nodes $V = \{1, \ldots p\}$ and edges $E = \{e_1, \ldots e_m\}$. In the special case of the

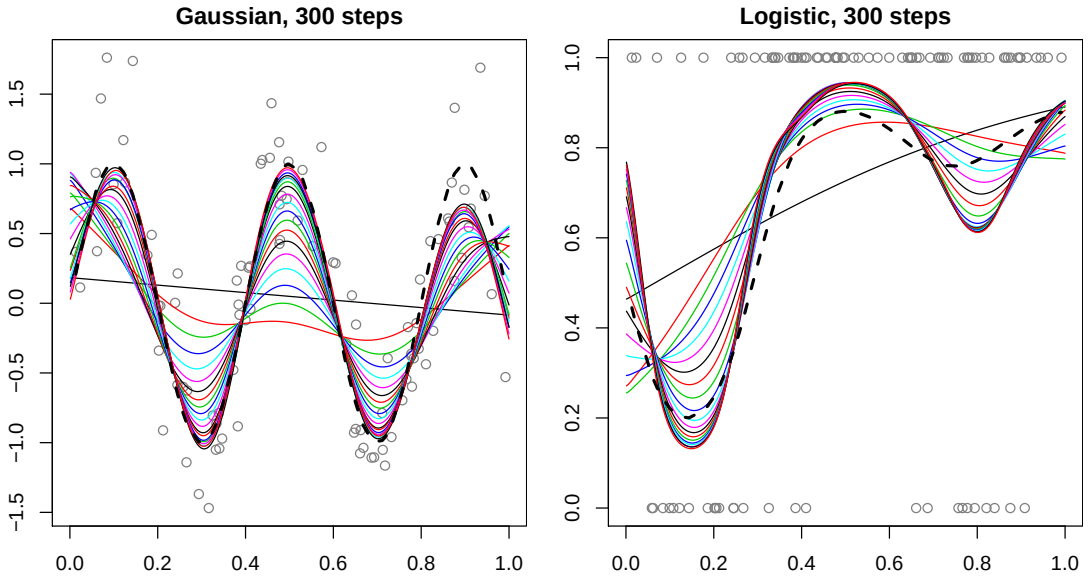


Figure 3: *Snapshots of the stagewise path for P-spline regularization problems, with continuous data in the left panel, and binary data in the right panel. In both examples, we use* $n = 100$ *points, and the true data generating curve is displayed as a thick dotted black line. The colored curves show the stagewise estimates over the first 300 path steps (plotted are every 15th estimate, for visibility).*

chain graph, wherein $E = \{\{1, 2\}, \{2, 3\}, \ldots \{p-1, p\}\}$, we have

$$D = \begin{bmatrix} -1 & 1 & 0 & \ldots & 0 & 0 \\ 0 & -1 & 1 & \ldots & 0 & 0 \\ \vdots & & & & & \\ 0 & 0 & 0 & \ldots & -1 & 1 \end{bmatrix},$$

so that $\|D\beta\|_1 = \sum_{j=1}^{p-1} |\beta_j - \beta_{j+1}|$. This regularization term encourages the ordered components of β to be piecewise constant, and problem (37) with this particular choice of D is usually called the *1-dimensional fused lasso* in the statistics literature (Tibshirani et al. 2005), or *1-dimensional total variation denoising* in signal processing (Rudin et al. 1992). In general, the edge incidence matrix $D \in \mathbb{R}^{m \times p}$ has rows corresponding to edges in E, and its ℓth row is

$$D_\ell = (0, \ldots \underset{\substack{\uparrow \\ i}}{-1}, \ldots \underset{\substack{\uparrow \\ j}}{1}, \ldots 0) \in \mathbb{R}^p,$$

provided that the ℓth edge is $e_\ell = \{i, j\}$. Hence $\|D\beta\|_1 = \sum_{\{i,j\} \in E} |\beta_i - \beta_j|$, a regularization term that encourages the components of β to be piecewise constant with respect to the structure defined by the graph G. Higher degrees of smoothness can be regularized in this framework as well, using *trend filtering* methods; see Kim et al. (2009) or Tibshirani (2014) for the 1-dimensional case, and Wang et al. (2015) for the more general case over arbitrary graphs.

Unfortunately the stagewise update in (6), under the regularizer $g(\beta) = \|D\beta\|_1$, is not computationally tractable. Computing this update is the same as solving a linear program, absent of any special structure in the presence of a generic matrix D. But we can make progress by studying the generalized lasso from the perspective of convex duality. Our jumping point for the dual is actually

the Lagrange form of problem (37), namely

$$\hat{\beta}(\lambda) \in \underset{\beta \in \mathbb{R}^p}{\operatorname{argmin}} \; f(\beta) + \lambda \|D\beta\|_1, \tag{38}$$

with $\lambda \geq 0$ now being the regularization parameter. The switch from (37) to (38) is justified because the two parametrizations admit identical solution paths. Following standard arguments in convex analysis, the dual problem of (38) can be written as

$$\hat{u}(\lambda) \in \underset{u \in \mathbb{R}^m}{\operatorname{argmin}} \; f^*(-D^T u) \;\; \text{subject to} \;\; \|u\|_\infty \leq \lambda, \tag{39}$$

with f^* denoting the convex conjugate of f. The primal and dual solutions satisfy the relationship

$$\nabla f(\hat{\beta}(\lambda)) + D^T \hat{u}(\lambda) = 0. \tag{40}$$

The general strategy is now to apply the stagewise algorithm to the dual problem (39) to produce an approximate dual solution path, and then convert this into an approximate primal solution path via (40). The stagewise procedure for (39) can be initialized with $\lambda_0 = 0$ and $u^{(0)} = 0$, and the form of the updates is described next. We assume that the conjugate function f^* is differentiable, which holds if f is strictly convex.

Lemma 5. *Applied to the problem* (39)*, the general stagewise procedure in Algorithm 2 repeats the updates* $u^{(k)} = u^{(k-1)} + \Delta$*, where*

$$\Delta_i = -\epsilon \cdot \begin{cases} 1 & \big[D \nabla f^*(-D^T u^{(k-1)})\big]_i < 0 \\ -1 & \big[D \nabla f^*(-D^T u^{(k-1)})\big]_i > 0 \\ 0 & \big[D \nabla f^*(-D^T u^{(k-1)})\big]_i = 0 \end{cases} \quad \textit{for } i = 1, \ldots m. \tag{41}$$

The proof follows from the duality of the ℓ_∞ and ℓ_1 norms, and the alternative representation in (15) for stagewise updates. Computation of Δ in (41), aside from evaluating the gradient ∇f^*, reduces to two matrix multiplications: one by D and one by D^T. In many cases (e.g., fused lasso and trend filtering problems), the matrix D is sparse, which makes this update step very cheap. To reiterate the dual strategy: we compute the dual estimates $u^{(k)}$, $k = 1, 2, 3, \ldots$ using the stagewise updates outlined above, and we compute primal estimates $\beta^{(k)}$, $k = 1, 2, 3, \ldots$ by solving for $\beta^{(k)}$ in the stationarity condition

$$\nabla f(\beta^{(k)}) + D^T u^{(k)} = 0, \tag{42}$$

for each k. The kth dual iterate $u^{(k)}$ is viewed as an approximate solution in (39) at $\lambda = \|u^{(k)}\|_\infty$, and the kth primal iterate $\beta^{(k)}$ an approximate solution in (37) at $t = \|D\beta^{(k)}\|_1$.

As pointed out by a referee of this paper, there is a key relationship between f and its conjugate f^* that simplifies the update direction in (41) considerably. At step k, observe that

$$\nabla f^*(-D^T u^{(k-1)}) = \nabla f^*(\nabla f(\beta^{(k-1)})) = \beta^{(k-1)}.$$

The first equality comes from the primal-dual relationship (40) at step $k-1$, and the second is due to the fact that $x = \nabla f^*(z) \iff z = \nabla f(x)$. As a result, the dual update $u^{(k)} = u^{(k-1)} + \Delta$ with Δ as in (41) can be written more succinctly as

$$u^{(k)} = u^{(k-1)} - \epsilon \cdot \operatorname{sign}(D\beta^{(k-1)}), \tag{43}$$

where $\operatorname{sign}(\cdot)$ is to be interpreted componentwise (with the convention $\operatorname{sign}(0) = 0$). Therefore, one can think of the dual stagewise strategy as alternating between computing a dual estimate $u^{(k)}$ as in (43), and computing a primal estimate $\beta^{(k)}$ by solving (42).

We note that, since the stagewise algorithm is being run through the dual, the estimates $\beta^{(k)}$, $k = 1, 2, 3, \ldots$ for generalized lasso problems differ from those in the other stagewise implementations encountered thus far, in that $\beta^{(k)}$, $k = 1, 2, 3, \ldots$ correspond to approximate solutions at *increasing* levels of regularization, as k increases. That is, the stagewise algorithm for problem (37) begins at the unregularized end of the path and iterates towards the fully regularized end, which is opposite to its usual direction.

A special case worth noting is that of Gaussian signal approximator problems, where the loss is $f(\beta) = \frac{1}{2}\|y - \beta\|_2^2$. For such problems, the primal-dual relationship in (42) reduces to

$$\beta^{(k)} = y - D^T u^{(k)},$$

for each k. This means that the initialization $u^{(0)} = 0$ and $\lambda_0 = 0$ in the dual is the same as $\beta^{(0)} = y$ and $t_0 = \|Dy\|_1$ in the primal. Furthermore, it means that the dual updates in (43) lead to primal updates that can be expressed directly as

$$\beta^{(k)} = \beta^{(k-1)} - \epsilon \cdot D^T \mathrm{sign}(D\beta^{(k-1)}). \tag{44}$$

From the pure primal perspective, therefore, the stagewise algorithm begins with the trivial unregularized estimate $\beta^{(0)} = y$, and to fit subsequent estimates in (44), it iteratively shrinks along directions opposite to the active rows of D. That is, if $D_\ell \beta^{(k-1)} > 0$ (where D_ℓ is the ℓth row of D), then the algorithm adds D_ℓ^T to $\beta^{(k-1)}$ in forming $\beta^{(k)}$, which shrinks $D_\ell \beta^{(k)}$ towards zero, as $D_\ell D_\ell^T > 0$ (recall that D_ℓ is a row vector). The case $D_\ell \beta^{(k-1)} < 0$ is similar. If $D_\ell \beta^{(k-1)} = 0$, then no shrinkage is applied along D_ℓ.

This story can be made more concrete for fused lasso problems, where D is the edge incidence matrix of a graph: here the update in (44) evaluates the differences across neighboring components of $\beta^{(k-1)}$, and for any nonzero difference, it shrinks the associated components towards each other to build $\beta^{(k)}$. The level of shrinkage is uniform across all active differences, as any two neighboring components move a constant amount ϵ towards each other.[6] This is a simple and natural iterative procedure for fitting piecewise constant estimates over graphs. For small examples using 1d and 2d grid graphs, see Appendix A.3.

4 Large examples and practical considerations

We compare the proposed general stagewise procedure to various alternatives, with respect to both computational and statistical performance, across the three of the four major regularization settings seen so far. The fourth setting is moved to the appendix, for space; see Appendix A.4. The current section specifically investigates large examples, at least relative to the small examples presented in Sections 1–3. Of course, one can surely find room to criticize our comparisons, e.g., with respect to a different tuning of the algorithm that computes exact solutions, a coarser grid of regularization parameter values over which it computes solutions, a different choice of algorithm completely, etc. We have tried to conduct fair comparisons in each problem setting, but we recognize that perfectly fair and exhaustive comparisons are near impossible. The message that we hope to convey is not that the stagewise algorithm is computationally superior to other algorithms in the problems we consider, but rather, that the stagewise algorithm is computationally competitive with the others, yet it is very simple, and capable of producing estimates of high statistical quality.

4.1 Group lasso regression

Overview. We examine two simulated high-dimensional group lasso regression problems. To compute group lasso solution paths, we used the `SGL` R package, available on the CRAN repository.

[6] This is assuming that D is the edge incidence matrix of an unweighted graph; with edge weights, the rows of D scale accordingly, and so the effective amounts of shrinkage in the stagewise algorithm scale accordingly too.

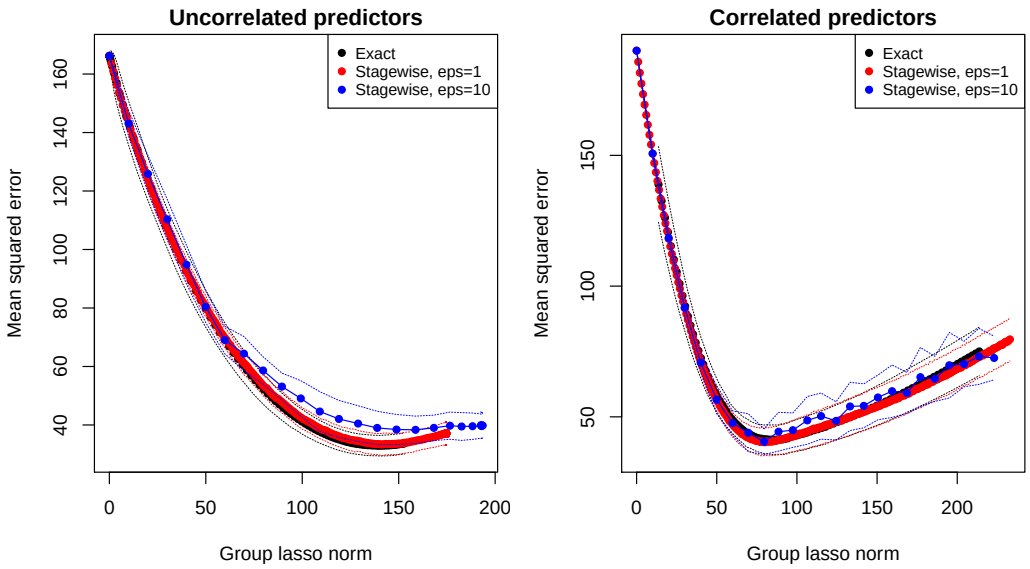
This package implements a block coordinate descent algorithm for solving the group lasso problem, where each block update itself applies accelerated proximal gradient descent (Simon et al. 2013). This idea is not complicated, but an efficient implementation of this algorithm requires care and attention to detail, such as backtracking line search for the proximal gradient step sizes. The stagewise algorithm, on the other hand, is very simple—in C++, the implementation is only about 50 lines of code. Refer to Section 3.1 for a description of the stagewise update steps. The algorithmics of the `SGL` package are also written in C++.

Examples and comparisons. In both problem setups, we used $n = 200$ observations, $p = 4000$ predictors, and $G = 100$ equal-sized groups (of size 40). The true coefficient vector $\beta^* \in \mathbb{R}^{4000}$ was defined to be group sparse, supported on only 4 groups, and the nonzero components were drawn independently from $N(0, 1)$. We generated observations $y \in \mathbb{R}^{200}$ by adding independent $N(0, \tau^2)$ noise to $X\beta^*$, where the predictor matrix $X \in \mathbb{R}^{200\times 4000}$ and noise level τ were chosen under two different setups. In the first, the entries of X were drawn independently from $N(0, 1)$, so that the predictors were uncorrelated (in the population); we also let $\tau = 6$. In the second, each row of X was drawn independently from a $N(0, \Sigma)$ distribution, where Σ had a block correlation structure. The covariance matrix Σ was defined so that each predictor variable had unit (population) variance, but (population) correlation $\rho = 0.85$ with 99 other predictors, each from a different group. Further, in this second setup, we used an elevated noise level $\tau = 10$.

Figure 4 shows a comparison of the group lasso and stagewise paths, from both computational and statistical perspectives. We fit group lasso solutions over 100 regularization parameter values (the `SGL` package started at the regularized end, and used warm starts). We also ran the stagewise algorithm in two modes: for 250 steps with $\epsilon = 1$, and for 25 steps with $\epsilon = 10$. The top row of Figure 4 asserts that, in both the uncorrelated and correlated problem setups, the mean squared errors of the stagewise fits $X\beta^{(k)}$ to the underlying mean $X\beta^*$ are quite competitive with those of the exact fits $X\hat{\beta}(t)$. In both plots, the red and black error curves, corresponding to the stagewise fits with $\epsilon = 1$ and the exact fits, respectively, lie directly on top of each other. It took less than 1 second to compute these stagewise fits, in either problem setup; meanwhile, it took about 10 times this long to compute the group lasso fits in the uncorrelated setup, and 100 times this long in the correlated setup. The stagewise algorithm with $\epsilon = 10$ took less than 0.1 seconds to compute a total of 25 estimates, and offers a slightly degraded but still surprisingly competitive mean squared error curve, in both the correlated and uncorrelated problem setups. Exact timings can be found in the middle row of Figure 4. The error curves and timings were all averaged over 10 draws of observations y from the uncorrelated or correlated simulation models (for fixed X, β^*); the timings were made on a desktop personal computer.

Though the exact and stagewise component paths typically appear quite similar in the uncorrelated problem setup, the same is not true for the correlated setup. The bottom row of Figure 4 displays an example of the two sets of component paths for one simulated draw of observations, under the correlated predictor model. The component paths of the group lasso solution, on the left, vary wildly with the regularization parameter; the stagewise paths, on the right, are much more stable. It is interesting to see that such different estimates can yield similar mean squared errors (as, recall, shown in the top row of Figure 4) but this is the nature of using correlated predictors in a regression problem.

Frank-Wolfe. We include a comparison to the Frank-Wolfe algorithm for computing group lasso solutions, across the same 100 regularization parameter values considered by the coordinate descent method. Recall that the updates from Frank-Wolfe share the same computational underpinnings as the stagewise ones, but are combined in a different manner; refer to Appendix A.1 for details. We implemented the Frank-Wolfe method for group lasso regression in C++, which starts at the largest regularization parameter value, and uses warm starts along the parameter sequence. The middle row of Figure 4 reports the Frank-Wolfe timings, averaged over 10 draws from the uncorrelated and



Algorithm timings		
Method	Uncorrelated case	Correlated case
Exact: coordinate descent, 100 solutions	9.08 (1.06)	78.64 (17.92)
Stagewise: $\epsilon = 1$, 250 estimates	0.93 (0.00)	0.94 (0.01)
Stagewise: $\epsilon = 10$, 25 estimates	0.09 (0.00)	0.10 (0.01)
Frank-Wolfe: within 1% of criterion value	67.73 (10.37)	92.91 (8.37)
Frank-Wolfe: within 1% of mean squared error	1.30 (0.56)	13.17 (26.26)

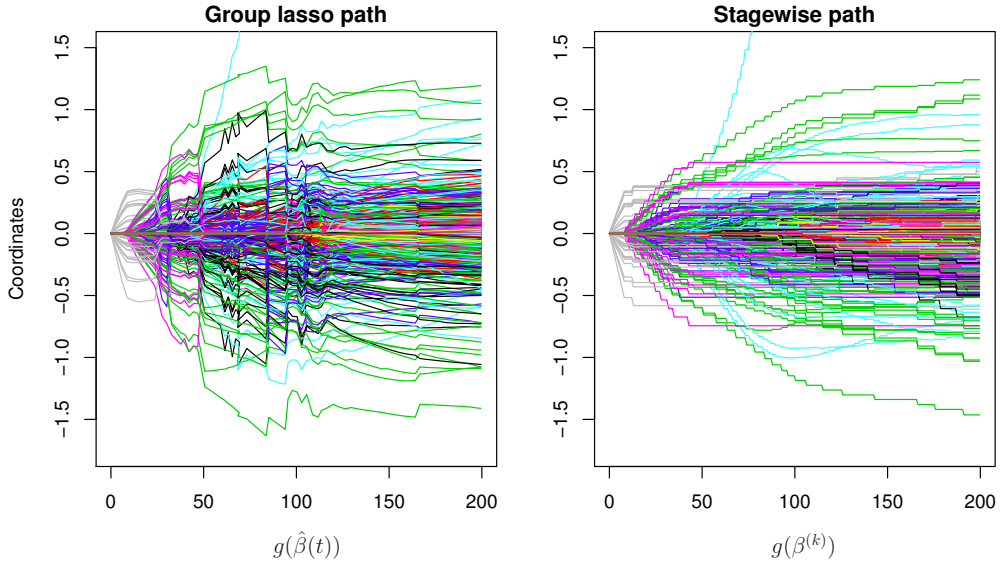


Figure 4: *Statistical and computational comparisons between group lasso solutions and corresponding estimates produced by the stagewise approach, when* $n = 200$, $p = 4000$. *The top row shows that stagewise estimates can achieve competitive mean squared errors to that of group lasso solutions, as computed by coordinate descent, under two different setups for the predictors in group lasso regression: uncorrelated and block correlated. (The curves were averaged over 10 simulations, with standard deviations denoted by dotted lines.) The middle table reports runtimes in seconds (averaged over 10 simulations, with standard deviations in parentheses) for the various algorithms considered, and shows that the stagewise algorithm represents a computationally attractive alternative to the* `SGL` *coordinate descent approach and the Frank-Wolfe algorithm. Lastly, the bottom row contrasts the group lasso and stagewise component paths, for one draw from the correlated predictors setup.*

correlated simulation models. We considered two schemes for termination of the algorithm, at each regularization parameter value t: the first terminates when

$$\|y - X\tilde{\beta}(t)\|_2^2 \leq 1.01 \cdot \|y - X\hat{\beta}(t)\|_2^2, \tag{45}$$

where $\tilde{\beta}(t)$ is the Frank-Wolfe iterate at t, and $\hat{\beta}(t)$ is the computed coordinate descent solution at t; the second terminates when

$$\|X\beta^* - X\tilde{\beta}(t)\|_2^2 \leq 1.01 \cdot \max\Big\{\|X\beta^* - X\hat{\beta}(t)\|_2^2, \|X\beta^* - X\beta^{(k_t)}\|_2^2\Big\}, \tag{46}$$

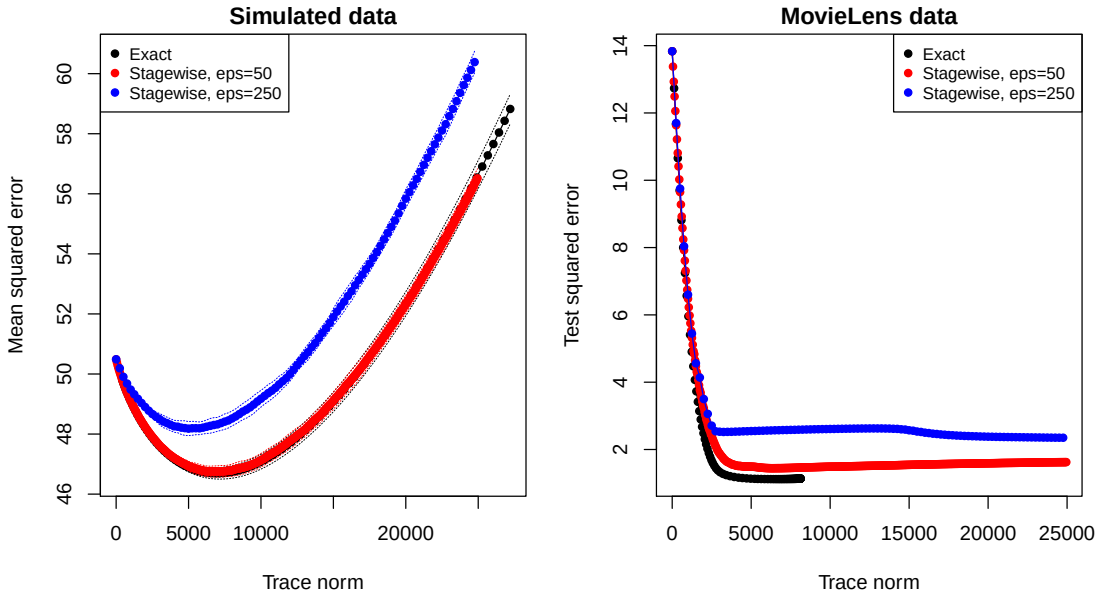
where $\beta^{(k_t)}$ is the imputed stagewise estimate at the parameter value t (computed by linear interpolation of the appropriate neighboring stagewise estimates). In other words, the first rule (45) stops when the Frank-Wolfe iterate is within 1% of the criterion value achieved by the coordinate descent solution, and the second rule (46) stops when the Frank-Wolfe iterate is within 1% of the mean squared error of either of the coordinate descent or stagewise fits. Using the first rule, the Frank-Wolfe algorithm took about 68 seconds to compute 100 solutions in the uncorrelated problem setup, and 93 seconds in the correlated problem setup. In terms of the total iteration count, this meant 18,627 Frank-Wolfe iterations in the uncorrelated case, and 25,579 in the correlated case; these numbers are meaningful, because, recall, one Frank-Wolfe iteration is (essentially) computationally equivalent to one stagewise iteration. We can see that Frank-Wolfe struggles here to compute solutions that match the accuracy of coordinate descent solutions, especially for large values of t—in fact, when we changed the factor of 1.01 to 1 in the stopping rule (45), the Frank-Wolfe algorithm converged far, far more slowly. (For this part, the coordinate descent solutions themselves were only computed to moderate accuracy; we used the default convergence threshold in the `SGL` package.) The results are more optimistic under the second stopping rule. Under this rule, the Frank-Wolfe algorithm ran in just over 1 second (274 iterations) in the uncorrelated setup, and about 13 seconds (3592 iterations) in the correlated setup. But this stopping rule represents an idealistic situation for Frank-Wolfe, and moreover, it cannot be realistically applied in practice, since it relies on the underlying mean $X\beta^*$.

4.2 Matrix completion

Overview. We consider two matrix completion examples, one simulated and one using real data. To compute solutions of the matrix completion problem, under trace norm regularization, we used the `softImpute` R package from CRAN, which implements proximal gradient descent (Mazumder et al. 2010). The proximal operator here requires a truncated singular value decomposition (SVD) of a matrix the same dimensions as the input (partially observed) matrix Y. SVD calculations are generally very expensive, but for this problem a partial SVD can be efficiently computed with clever schemes based on bidiagonalization or alternating least squares. The `softImpute` package uses the latter scheme to compute a truncated SVD, and though this does provide a substantial improvement over the naive method of computing a full SVD, it is still far from cheap. The partial SVD computation via alternating least squares scales roughly quadratically with the rank of the sought solution, and this must be repeated for every iteration taken by the algorithm until convergence.

In comparison, the stagewise steps for the matrix completion problem require only the top left and right singular vectors of a matrix the same size as the input Y. Refer back to Section 3.3 for an explanation. To emphasize the differences between the two methods: the proximal gradient descent algorithm of `softImpute`, at each regularization parameter value t of interest, must iteratively compute a partial SVD until converging on the desired solution; the stagewise algorithm computes a single pair of left and right singular vectors, to form one estimate at one parameter value t, and then moves on to the next value of t. For the following examples, we used a simple R implementation of the stagewise algorithm; the computational core of the `softImpute` package is also written in R.

Examples and comparisons. In the first example, we simulated an underlying low-rank matrix $B^* \in \mathbb{R}^{500\times 500}$, of rank 50, by letting $B^* = UU^T$, where $U \in \mathbb{R}^{500\times 50}$ had independent $N(0,1)$ entries. We then added $N(0,20)$ noise, and discarded 40% of the entries, to form the input matrix $Y \in \mathbb{R}^{500\times 500}$ (so that Y was 60% observed). We ran `softImpute` at 100 regularization parameter values (starting at the regularized end, and using warm starts), and we ran two different versions of the stagewise algorithm: one with $\epsilon = 50$, for 500 steps, and one with $\epsilon = 250$, for 100 steps. The left plot in Figure 5 shows the mean squared error curves of the resulting estimates, averaged over 10 draws of the input matrix Y from the above prescription (with B^* fixed). We can see that the stagewise estimates, with $\epsilon = 50$, trace out an essentially identical mean squared error curve to that from the exact solutions. We can also see that, curiously, the larger step size $\epsilon = 250$ leads to suboptimal performance in stagewise estimation, as measured by mean squared error. This is unlike the previous group lasso setting, in which a larger step size still yielded basically the same performance (albeit slightly noisier mean squared error curves).



Algorithm timings		
Method	Simulated data	MovieLens data
Exact: proximal gradient, 100 solutions	60.20 (1.45)	334.67
Stagewise: $\epsilon = 50$, 500 estimates	92.92 (2.42)	107.66
Stagewise: $\epsilon = 250$, 100 estimates	18.26 (0.98)	21.22
Frank-Wolfe: within 1% of criterion value	989.77 (19.88)	-
Frank-Wolfe: within 1% of mean squared error	154.06 (10.76)	-

Figure 5: *Comparisons between exact and stagewise estimates for matrix completion problems. The top left plot shows mean squared error curves for a simulated example of a 40% observed,* 500×500 *input matrix, and the right shows the same for the MovieLens data, where the input is 6% observed and* 943×1682*. (The error curves in the left plot were averaged over 10 repetitions, and standard deviations are drawn as dotted lines.) The stagewise estimates with* $\epsilon = 50$ *are competitive in both cases. The bottom table gives the runtimes of* `softImpute` *proximal gradient descent, stagewise, and the Frank-Wolfe algorithm. (Timings for the simulated case were averaged over 10 repetitions, with standard deviations in parentheses; Frank-Wolfe was not run on the MovieLens example.)*

In this simulated example, the proximal gradient descent method implemented by `softImpute`

took an average of 206 iterations to compute 100 solutions across 100 values of the regularization parameter (averaged over the 10 repetitions of the observation matrix Y). This means an average of just 2.06 iterations per solution—quite rapid convergence behavior for a first-order method like proximal gradient descent. (Note: we used the default convergence threshold for `softImpute`, which is only moderately small.) The stagewise algorithms, using step sizes $\epsilon = 50$ and $\epsilon = 250$, ran for 500 and 100 iterations, respectively. As explained, the two types of iterations here are different in nature. Each iteration of proximal gradient descent computes a truncated SVD, which is of roughly quadratic complexity in the rank of current solution, and therefore becomes more expensive as we progress down the regularization path; each stagewise iteration computes a single pair of left and right singular vectors, which has the same cost throughout the path, independent of the rank of the current estimate. The bottom row of Figure 5 is a table containing the running times of these two methods (averaged over 10 draws of Y, and recorded on a desktop computer). We see that proximal gradient descent spent an average of about 60 seconds to compute 100 solutions, i.e., 0.6 seconds per solution. The stagewise algorithm with $\epsilon = 50$ took an average of about 93 seconds for 500 steps, and the algorithm with $\epsilon = 250$ an average of 18 seconds for 100 steps, with both translate into about 0.18 seconds per estimate. The speedy time of 0.6 seconds per estimate of `softImpute` is explained by two factors: fast iterations (using the impressive, custom alternating least squares routine developed by the package authors to compute partial SVDs), and few iterations needed per solution (recall, only an average of 2.06 per solution in this example). The 0.18 seconds per stagewise iteration reflects the runtime of computing leading left and right singular vectors with R's standard `svd` function, as our implementation somewhat naively does (it does not take advantage of sparsity in any way). This naive stagewise implementation works just fine for moderate matrix sizes, as in the current example. But for larger matrix sizes (and higher levels of missingness), we see significant improvements when we use a more specialized routine for computing the top singular vectors. We also see a bigger separation in the costs per estimate with stagewise and proximal gradient descent. This is discussed next.

The second example is based on the MovieLens data set (collected by the GroupLens Research Project at the University of Minnesota, see `http://grouplens.org/datasets/movielens/`). We examined a subset of the full data set, with 100,000 ratings from $m = 943$ users on $n = 1682$ movies (hence the input matrix $Y \in \mathbb{R}^{943 \times 1682}$ was approximately 6% observed). We used an 80%/20% split of these ratings for training and testing, respectively; i.e., we computed matrix completion estimates using the first 80% of the ratings, and evaluated test errors on the held out 20% of the ratings. For the estimates, we ran `softImpute` over 100 values of the regularization parameter (starting at the regularized end, using warm starts), and stagewise with $\epsilon = 50$ for 500 steps, as well as with $\epsilon = 250$ for 100 steps. The right plot of Figure 5 shows the test error curves from each of these methods. The stagewise estimates computed with $\epsilon = 50$ and the exact solutions perform quite similarly, with the exact solutions having a slight advantage as the trace norm exceeds about 2500. The stagewise error curve when $\epsilon = 250$ begins by dropping off strongly just like the other two curves, but then it flattens out too early, while the other two continue descending. (We note that, for step sizes larger than $\epsilon = 250$, the test error curve stops decreasing even earlier, and for step sizes smaller than $\epsilon = 50$, the error curve reaches a slightly lower minimum, in line with that of the exact solution. This type of behavior is reminiscent of boosting algorithms.)

In terms of computation, the proximal gradient descent algorithm used a total of 1220 iterations to compute 100 solutions in the MovieLens example, or an average of 122 iterations per solution. This is much more than the 2.06 seconds per iteration as in the previous simulated example, and it explains the longer total runtime of about 335 seconds, i.e., the longer total time of 33.5 seconds per solution. The stagewise algorithms ran, by construction, for 500 and 100 steps and took about 108 and 21 seconds, respectively, i.e., an average of 0.21 seconds per estimate. To compute the leading left and right singular vectors in each stagewise step here, we used the `rARPACK` R package from CRAN, which accomodates sparse matrices. This was highly beneficial because the gradient $\nabla f(B^{(k-1)})$ at each stagewise step was very sparse (about 6% entries of its were nonzero, since Y

was about 6% observed).

Frank-Wolfe. We now compare the Frank-Wolfe algorithm for computing matrix completion solutions, over the same 100 regularization parameter values used by `softImpute`. Each Frank-Wolfe iteration computes a single pair of left and right top singular vectors, just like stagewise iterations; see Appendix A.1 for a general description of the Frank-Wolfe method (or Jaggi & Sulovsky (2010) for a study of Frank-Wolfe for trace norm regularization problems in particular). We implemented the Frank-Wolfe algorithm for matrix completion in R, which starts at the regularized end of the path, and uses warm starts at each regularization parameter value. The timings for the Frank-Wolfe method, run on the simulated example, are given in the table in Figure 5 (we did not run it on the MovieLens example). As before, in the group lasso setting, we considered two different stopping rules for Frank-Wolfe, applied at each regularization parameter value t: the first stops when the achieved criterion value is within 1% of that achieved by the proximal gradient descent approach in `softImpute`, and the second stops when the achieved mean squared error is within 1% of either that of `softImpute` or stagewise. In either case, we cap the maximum number of iterations at 100, at each parameter value t.

Under the first stopping rule, the Frank-Wolfe algorithm required an average of 5847 iterations to compute 100 solutions (averaged over 10 draws of the input matrix Y); furthermore, this total was calculated under the limit of 100 maximum iterations per solution, and the algorithm met this limit at each one of the largest 50 regularization parameter values t. Recall that each one of these Frank-Wolfe iterations is computationally equivalent to a stagewise iteration. Accordingly, 500 steps of the stagewise algorithm, with $\epsilon = 50$, ran in about an order of magnitude less time—93 seconds versus 990 seconds. The message is that the Frank-Wolfe algorithm experiences serious difficulty in producing solutions at a level of accuracy close to that of proximal gradient descent, especially for lower levels of regularization. Using the second stopping rule, Frank-Wolfe ran much faster, and computed 100 solutions in about 997 iterations, or 154 seconds. However, there are two important points to stress. First, this rule is not generally available in practice, as it depends on performance measured with respect to the true matrix B^*. Second, the termination behavior under this rule is actually somewhat misleading, because once the mean squared error curve begins to rise (in the left plot of Figure 5, after about $t = 7000$ in trace norm), the second rule will always cause Frank-Wolfe with warm starts to trivially terminate in 1 iteration. Indeed, in the simulated data example, the Frank-Wolfe algorithm using this rule took about 22 iterations per solution before $t = 7000$, and trivially 1 iteration per solution after this point.

4.3 Image denoising

Overview. We study the image denoising problem, cast as a generalized lasso problem with Gaussian signal approximator loss, and 2d fused lasso or 2d total variation regularization (meaning that the underlying graph is a 2d grid). To compute exact solutions of this problem, we applied a direct (noniterative) algorithm of Chambolle & Darbon (2009), that reduces this problem to sequence of maximum flow problems. The "parametric" maximum flow approach taken by these authors is both very elegant and highly specialized. To the best of our knowledge, their algorithm is one of the fastest existing algorithms for 2d fused lasso problems (more generally, fused lasso problems over graphs). For the simulations in this section we relied on a fast C++ implementation provided by the authors (see `http://www.cmap.polytechnique.fr/~antonin/software/`), which totals close to 1000 lines of code. The stagewise algorithm is almost trivially simple in comparison, as our own C++ implementation requires only about 50 lines of code. For the 2d fused lasso regularizer, the stagewise update steps reduce to sparse matrix multiplications; refer to Section 3.5 for details.

Examples and comparisons. We inspect two image denoising examples. For the first, we constructed a 300×200 image to have piecewise constant levels, and added independent $N(0, 1)$ noise

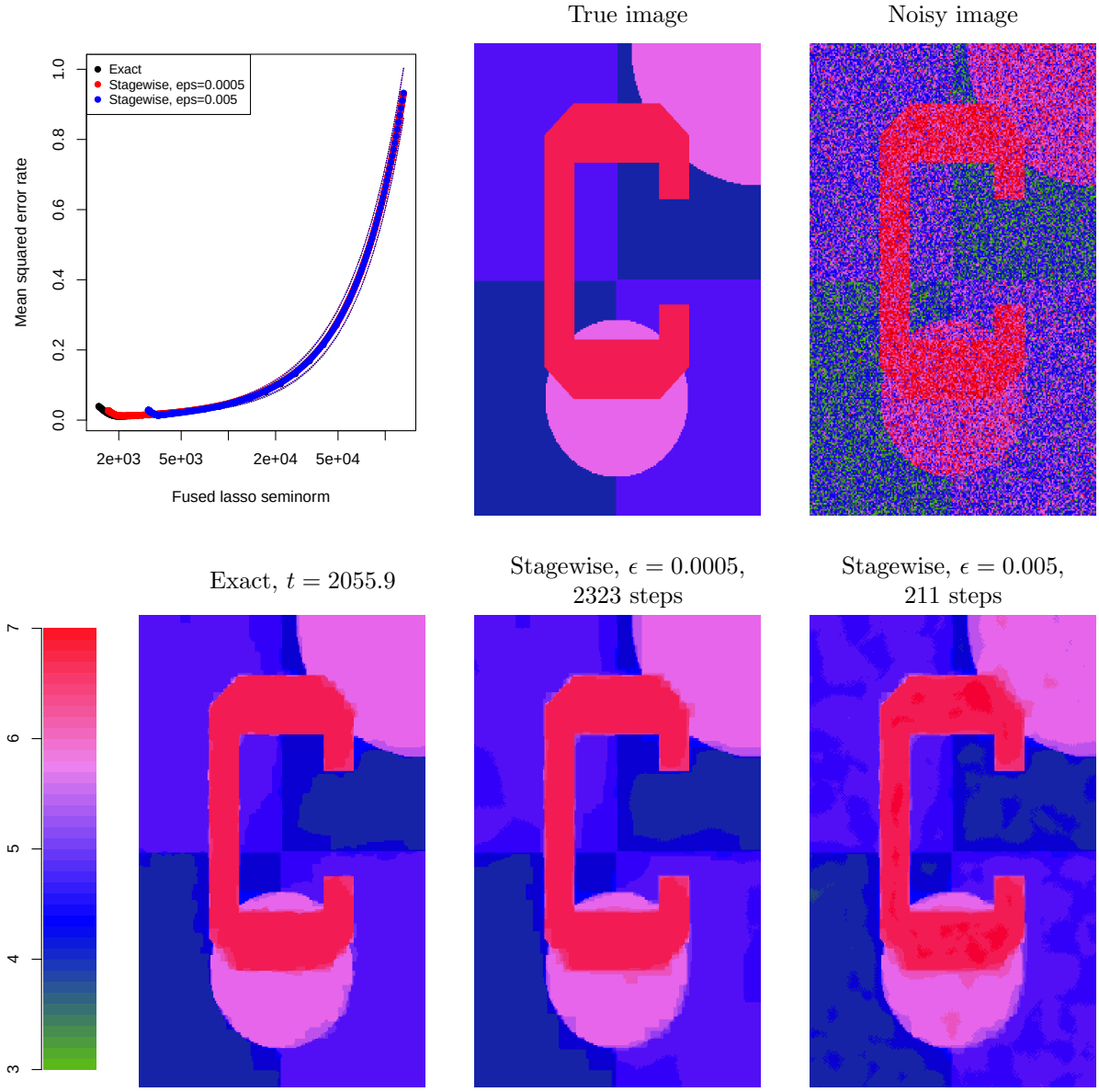
to the level of each pixel. Both this true underlying image and its noisy version are displayed in Figure 6. We then ran the parametric max flow approach of Chambolle & Darbon (2009), to compute exact 2d fused lasso solutions, at 100 values of the regularization parameter. (This algorithm is direct and does not take warm starts, so each instance was solved separately.) We also ran the stagewise method in two modes: for 6000 steps with $\epsilon = 0.0005$, and for 500 steps with $\epsilon = 0.005$. The mean squared error curves for each method are shown in the top left corner of Figure 6, and timings are given in the bottom table. (All results here have been averaged over 10 draws of the noisy image, and the timings were recorded on a desktop computer.) We can see that the stagewise estimates, both with $\epsilon = 0.0005$ and $\epsilon = 0.005$, perform comparably to the exact solutions in terms of mean squared error, though the estimates under the smaller step size fare slightly better towards the more regularized end of the path. The 6000 stagewise estimates using $\epsilon = 0.0005$ took about 15 seconds to compute, and the 500 stagewise estimates using $\epsilon = 0.005$ took roughly 1.5 seconds. The max flow approach required an average of about 110 seconds to compute 100 solutions, with the majority of computation time spent on solutions at higher levels of regularization (which, here, correspond to lower mean squared errors). Finally, the estimate from each method that minimized mean squared error is also plotted in Figure 6; all look very similar and do a visually reasonable job of recovering the underlying image. That the stagewise approach can deliver such high-quality denoised images with simple, cheap iterations is both fortuituous and surprising.

The second example considers the stagewise algorithm for a larger-scale image denoising task, based on a real 640×480 image, of Lake Pukaki in front of Mount Cook, New Zealand. We worked with each color channel—red, green, blue—separately, and the pixel values were scaled to lie between 0 and 1. For each of these three images, we added independent $N(0, 0.5)$ noise to the pixel values, and ran the stagewise algorithm with $\epsilon = 0.005$ for 650 steps. We chose this number of steps because the achieved mean squared error (averaged over the three color channels) roughly began to rise after this point. We then recombined the three denoised images—on the red, green, blue color channels—to form a single image. See Figure 7. Visually, the reconstructed image is remarkably close to the original one, especially considering the input noisy image on which it is computed. The stagewise algorithm took a total of around 21 seconds to produce this result; recall, though, that in this time it actually produced $650 \times 3 = 1950$ fused lasso estimates (650 steps in three different image denoising tasks, one for each color).

4.4 Choice of step size

We discuss a main practical issue when running the stagewise algorithm: choice of the step size ϵ. Of course, when ϵ is too small, the algorithm is less efficient, and when ϵ is too large, the stagewise estimates can fail to span the full regularization path (or a sizeable portion of it). Our heuristic suggestion therefore is to start with a large step size ϵ, and plot the progress of the achieved loss $f(x^{(k)})$ and regularizer $g(x^{(k)})$ function values across steps $k = 1, 2, 3, \ldots$ of the algorithm. With a proper choice of ϵ, note that we should see $f(x^{(k)})$ monotone decreasing with k, as well as $g(x^{(k)})$ monotone increasing with k (this is true of $f(\hat{x}(t))$ and $g(\hat{x}(t))$ as we increase the regularization parameter t, in the exact solution computation). If ϵ is too large, then it seems to be the tendency in practice that the achieved values $f(x^{(k)})$ and $g(x^{(k)})$, $k = 1, 2, 3, \ldots$ stop their monotone progress at some point, and alternate back and forth. Figure 8 illustrates this behavior. Once encountered, an appropriate response would be decrease ϵ (say, halve it), and continue the stagewise algorithm from the last step before this alternating pattern surfaced.

The heuristic guideline above attempts to produce the largest step size ϵ that still produces an expansive regularization path of stagewise estimates. This ignores the subtlety that a larger choice ϵ may offer suboptimal statistical performance, even if the corresponding estimates span the full path. This was seen in some examples of Section 4 (e.g., matrix completion, in Figure 5), but not in others (e.g., group lasso regression, in Figure 4). The issue of tuning ϵ for optimal statistical performance is more complex and problem dependent. Although it is clearly important, we do not



Algorithm timings	
Method	Runtime
Exact: maximum flow, 100 solutions	109.04 (6.21)
Stagewise: $\epsilon = 0.0025$, 6000 estimates	15.11 (0.18)
Stagewise: $\epsilon = 0.25$, 500 estimates	1.26 (0.02)

Figure 6: *Comparisons between exact 2d fused lasso solutions and stagewise estimates on a synthetic image denoising example. The true underlying* 300×200 *image is displayed in the middle of the top row. (A color scale is applied for visualization purposes, see the left end of the bottom row.) Over 10 noisy pertubations of this underlying image, with one such example shown in the right plot of the top row, we compare averaged mean squared errors of the exact solutions and stagewise estimates, in the left plot of the top row. Average timings for these methods are given in the bottom table. (Standard deviations are denoted by dotted lines in the error plots, and are in parentheses in the table.) The stagewise estimates have competitive mean squared errors and are fast to compute. The bottom row of plots shows the optimal image (i.e., that minimizing mean squared error) from each method, based on the single noisy image in the top right.*

Original image:

Noisy version:

Stagewise, $\epsilon = 0.001$, 650 steps:

(computed in 21.34 seconds)

Figure 7: *A more realistic image denoising example using stagewise. We began with a* 640×480 *photograph of Lake Pukaki and Mount Cook, in New Zealand, shown at the top. Working with each color channel separately, we added noise to form the middle noisy image, and ran the stagewise algorithm to eventually yield the bottom image, a nice reconstruction.*

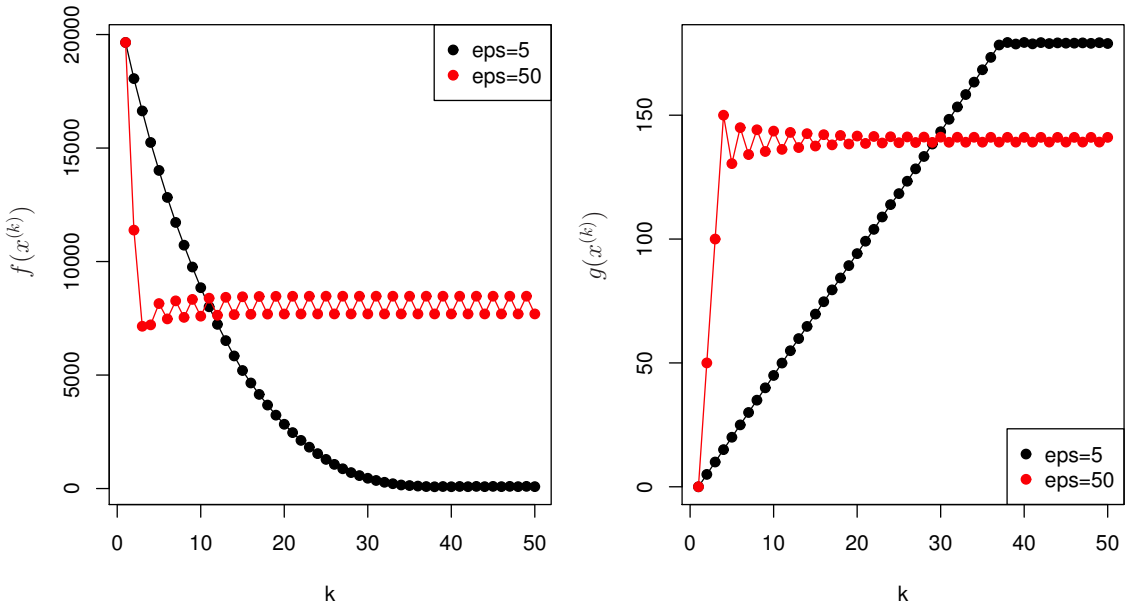


Figure 8: *An example displaying a common tendency of stagewise estimates under a choice of step size ϵ that is too large. We used the group lasso regression data setup from Figure 4 (uncorrelated case). Both the achieved loss $f(x^{(k)})$ (left plot) and regularizer $g(x^{(k)})$ (right plot) function values should be monotonic across steps $k = 1, 2, 3, \ldots$. We see that for the larger step size $\epsilon = 50$ (in red), progress halts and an alternating pattern begins, with both sequences; for the smaller step size $\epsilon = 5$ (in black), progress continues all the way until the end of the path.*

study this task in the current paper. We mention the (somewhat obvious) point that strategies like cross-validation (if applicable, in the given problem setting) could be helpful here.

5 Suboptimality bounds for stagewise estimates

5.1 General stagewise suboptimality

We present a suboptimality bound for estimates produced by the general stagewise algorithm, restricting our attention to a norm regularizer g. The following result makes use of the dual norm g^* of g which, recall, is defined as $g^*(x) = \max_{g(z) \leq 1} x^T z$. Its proof is based on recursively tracking a duality gap for the general problem (4), and is deferred until Appendix A.5.

Theorem 1. *Consider the general problem (4), assuming that f is differentiable and convex, and g is a norm. Assume also that ∇f is Lipschitz with respect to the pair g^*, g with constant L, i.e.,*

$$g^*\big(\nabla f(x) - \nabla f(y)\big) \leq L \cdot g(x - y), \quad \text{all } x, y.$$

Fix a regularization parameter value t of interest, and consider running the general stagewise algorithm, Algorithm 2, from $x^{(0)} = \hat{x}(t_0)$, a solution in (4) at a parameter value $t_0 \leq t$. Suppose that we run the algorithm for k steps, with step size ϵ, such that $t_k = t_0 + k\epsilon = t$. The resulting stagewise estimate $x^{(k)}$ satisfies

$$f(x^{(k)}) - f(\hat{x}(t)) \leq L(t^2 - t_0^2) + L(t - t_0)\epsilon.$$

Therefore, if we consider the limiting stagewise estimate at the parameter value t, denoted by $\tilde{x}(t)$, as the step size $\epsilon \to 0$, then such an estimate satisfies

$$f(\tilde{x}(t)) - f(\hat{x}(t)) \leq L(t^2 - t_0^2).$$

Remark 1. In the theorem, the kth stagewise estimate $x^{(k)}$ is taken to be an approximate solution at the static regularization parameter value $t_k = t_0 + k\epsilon$, not at the dynamic value $t_k = g(x^{(k)})$, as we have been considering so far. It is easy to see that with the static choice $t_k = t_0 + k\epsilon$, we have $g(x^{(k)}) \leq t_k$, so that $x^{(k)}$ is still feasible at the parameter t_k. Furthermore, this choice simplifies the analysis, and would also simplify running the algorithm in practice (when g is expensive to compute, e.g., in the trace norm setting).

Remark 2. The assumptions that f is differentiable and that its gradient ∇f is Lipschitz continuous are fairly standard in the analysis of optimization algorithms; usually the Lipschitz assumption is made with respect to a prespecified pair of primal and dual norms, but here instead we utilize the pair naturally suggested by the problem (4), namely, g, g^*. For example, in the least squares setting, $f(\beta) = \frac{1}{2}\|y - X\beta\|_2^2$, with an arbitrary norm g as the regularizer, the Lipschitz constant of ∇f is

$$L = \max_{u \neq 0} \frac{g^*(X^T X u)}{g(u)},$$

which we might write as $L = \|X^T X\|_{g,g^*}$ in the spirit of matrix norms.

Remark 3. The theorem can be extended to the case when g is a seminorm regularizer. As written, the Lipschitz constant L would be infinite if g has a nontrivial null space N_g that overlaps with ∇f, as made precise in (17). However, we could g^* redefine as

$$g^*(x) = \max_{z \in N_g^\perp,\, g(z) \leq 1} x^T z,$$

and one can then check that, under the same conditions, the proof of Theorem 1 goes through just as before, but now the bounds apply to the modified stagewise estimates in (18), (19).

5.2 Shrunken stagewise framework

For reasons that will become apparent, we introduce a shrunken version of the stagewise estimates.

Algorithm 3 (Shrunken stagewise procedure).

Fix $\epsilon > 0$, $\alpha \in (0, 1)$, $t_0 \in \mathbb{R}$. Set $x^{(0)} = \hat{x}(t_0)$, a solution in (4) at $t = t_0$. Repeat, for $k = 1, 2, 3, \ldots$,

$$x^{(k)} = \alpha x^{(k-1)} + \Delta, \tag{47}$$

$$\textit{where } \Delta \in \operatorname*{argmin}_{z \in \mathbb{R}^n} \langle \nabla f(x^{(k-1)}), z \rangle \;\text{ subject to }\; g(z) \leq \epsilon. \tag{48}$$

The only difference between Algorithm 3 and the existing stagewise proposal in Algorithm 2 is that the update step in (47) shrinks the current iterate $x^{(k-1)}$ by a constant amount $\alpha < 1$, before adding the direction Δ. Note that in the case of unbounded stagewise updates, we would replace (48) by the subspace constrained version (19), as explained in Section 2.3.

Before we give examples or theory, we motivate the study of the shrunken stagewise algorithm from a conceptual point of view. It helps to think about lasso regression in particular, with $f(\beta) = \frac{1}{2}\|y - X\beta\|_2^2$ and $g(\beta) = \|\beta\|_1$. Recall that in this case, the general stagewise procedure reduces to classical forward stagewise regression, in Algorithm 1. A step k, forward stagewise updates the component i of the estimate $\beta^{(k-1)}$ such that the variable X_i has the largest absolute inner product with the residual $y - X\beta^{(k-1)}$; further, it moves $\beta_i^{(k-1)}$ in a direction given by the sign of this inner product. It is intuitively clear why such a procedure generally yields monotone component paths: if X_i has a large positive inner product with the residual, and we add a small amount ϵ to the ith coefficient, then in the next step, X_i will still have a large positive inner product with the residual. This inner product will have been slightly decremented due to the change in ith coefficient, but we will continue to increment the ith coefficient by ϵ (decrement the ith inner product) until another

variable attains a comparable inner product with the residual. In other words, the ith component path computed by forward stagewise will increase monotonically, and eventually flatten out.

So how does nonmonoticity occur in stagewise paths? Keeping with the above thought experiment, in order for the ith coefficient path to decrease at some point, the variable X_i must achieve a *negative* inner product with the residual, and this must be largest in magnitude compared to the inner products from all other variables. Given that X_i had a large positive inner product with the residual in previous iterations, this seems highly unlikely, especially in a high-dimensional setting with many variables in total. But we know from many examples that the components of the exact lasso solution path can exhibit many nonmonoticities, even very early on in the regularization path, and even in high-dimensional settings. To recover the exact path with a stagewise-like algorithm, therefore, some change needs to be made to counteract the momentum gathered over successive updates. Zhao & Yu (2007) do just this, as discussed in the introduction, by adding an explicit backward step to the stagewise routine in which coefficients are driven towards zero as long as this decreases the loss by a significant amount.

An arguably simpler way to achieve a roughly similar effect is to shrink all coefficients towards zero at each step. This is what is done by the shrunken stagewise method, in Algorithm 3, via the parameter $\alpha < 1$. In shrunken stagewise for lasso regression, the importance of each variable wanes over steps of the algorithm. Thus, in the absence of attention from the stagewise update mechanism, a coefficient path slides towards zero, instead of leveling off; for a coefficient path to depart from zero, or even remain at a constant level, it must regain the attention of the update mechanism by repeatedly achieving the maximal absolute inner product. This actually represents a fairly different philosophy from the pure stagewise approach (with $\alpha = 1$) and the two can be crudely constrasted as follows: pure stagewise keeps coefficients at constant levels, unless there is good reason to move them away from zero; shrunken stagewise drives coefficients to zero, unless there is good reason to keep them on their current trajectories.

We give a small example of shrunken stagewise applied to lasso regression, with $n = 20$ observations and $p = 10$ variables. The rows of the predictor matrix $X \in \mathbb{R}^{20\times 10}$ were drawn independently from a Gaussian distribution with mean zero, and a covariance matrix having unit diagonals and constant off-diagonals $\rho = 0.8$. The underlying coefficient vector $\beta^* \in \mathbb{R}^{10}$ had dense support, with all entries drawn from $N(0, 1)$, and the observations y were formed by adding independent $N(0, 1)$ noise to $X\beta^*$. Figure 9 shows the exact lasso solution path on the left panel, the stagewise path in the middle panel, and the shrunkage stagewise path on the right. We can see that, at various points, components of the exact lasso path become nonmonotone, and as expected, the corresponding the stagewise component paths ignore this trend and level out. The shrunken stagewise component paths pick up on the nonmonoticities and actually mimick the exact ones quite closely. We note that the stagewise and shrunken stagewise algorithms were not run here for efficiency, but were run at fine resolution to reveal their limiting behaviors; both used a small step size $\epsilon = 0.0001$, and the latter used a shrinkage factor $\alpha = \epsilon/10$. The two required 100,000 and 500,000 steps, respectively.

To be upfront, we remark that the shrunken stagewise method is *not a computationally efficient approach*, and we do not advocate its use in practice. The stagewise algorithm in the above example could have been run, e.g., with $\epsilon = 0.01$ and for 100 steps, and this would have yielded a sequence of estimates with effectively the same pattern. But to capture the nonmonotonicities present in the exact solution path, larger step sizes do not suffice for shrunken stagewise, and the algorithm needs to be run with $\epsilon = 0.0001$ and for 500,000 steps—this is clearly not desirable for such a small example with $n = 20$ and $p = 10$, and it does not bode well for scalability. We will see in what follows that the shrunken stagewise estimates provide a bridge between pure stagewise estimates and exact solutions in the general convex regularization problem (4). Hence we view the shrunken stagewise estimates as interesting and worthwhile *because* they provides this connection.

The main reason we choose to study the shrinkage strategy in Algorithm 3, as opposed to, say, backward steps, is that the shrinkage approach applies outside of the lasso regularization setting; as far as we can tell, there is no natural analog of backwards steps beyond the sparse setting. In

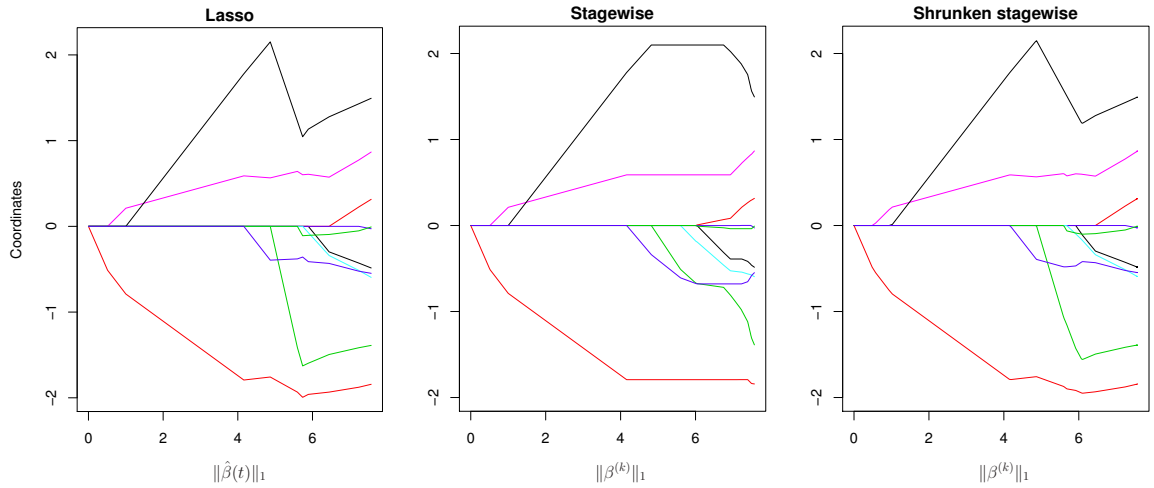


Figure 9: *Exact, stagewise, and shrunken stagewise paths for a small lasso regression problem with* $n = 20$ *observations, and* $p = 10$ *correlated predictors. When components of the lasso solution path become nonmonotone (e.g., top black path, and bottom red path), the corresponding stagewise ones are more stable and remain at a constant level, but shrunken stagewise matches the nonmonotonicities.*

fact, in the general problem setup, the shrinkage factor α in Algorithm 3 somewhat roughly mirrors what is done by Frank-Wolfe (this is really a different strategy, but still, it is one that computes exact solutions; compare equations (48) and (51) from Appendix A.1). A general interpretation of the shrinkage operation in (48) is that it lessens the dependence of the stagewise estimates on the computed history, i.e., decreases the stability of the computed stagewise component paths, and implicitly allows for more weight to be placed on the local update directions. Empirical examples with, e.g., group lasso regression or matrix completion confirm that shrunken stagewise estimates can be tuned to track the exact solution path even when the pure stagewise path deviates from it. We do not examine these cases here but instead turn to theoretical development.

5.3 Shrunken stagewise suboptimality

As in Section 5.1, we assume that g is a norm, and write g^* for its dual norm. We also consider the kth shrunken stagewise estimate $x^{(k)}$ as an approximate solution in the general problem (4) at a static value of the regularization parameter, defined recursively as $t_k = \alpha t_{k-1} + \epsilon$. A straightforward inductive argument shows that $g(x^{(k)}) \leq t_k$, i.e., the estimate $x^{(k)}$ is feasible for the problem (4) at $t = t_k$. Under this setup, the same limiting suboptimality bound as in Theorem 1 can be established for the shrunken stagewise estimates. For the sake of space, we do not present this result. Instead we show that, under additional conditions, the shrunken stagewise estimates overcome the stability inherent to stagewise, and achieve the idealized behavior suggested by Figure 9, i.e., they converge to exact solutions along the path. See Appendix A.6 for the proof.

Theorem 2. *Consider the general problem* (4)*. Assume, as in Theorem 1, that the loss function* f *is differentiable and convex, the regularizer* g *is a norm, and* ∇f *is Lipschitz with respect to* g^*, g*, having Lipschitz constant* L*. Fix a parameter value* t*, and consider running the shrunken stagewise algorithm, Algorithm 3, from* $x^{(0)} = \hat{x}(t_0)$*, a solution in* (4) *at a parameter value* $t_0 \leq t$*. Consider the limiting estimate* $\tilde{x}(t)$ *at the parameter value* t*, as both* $\epsilon \to 0$ *and* $\alpha \to 1$*. Suppose that*

$$\frac{1-\alpha}{\epsilon} \to 0 \quad \textit{and} \quad \frac{1-\alpha}{\epsilon^2} \to \infty.$$

Let $k = k(\epsilon, \alpha)$ denote the number of steps taken by the shrunken stagewise algorithm to reach the parameter value $t_k = t$; note that $k \to \infty$ as $\epsilon \to 0$, $\alpha \to 1$. Define the effective Lagrange parameters $\lambda_i = g^*(\nabla f(x^{(i)}))$, $i = 1, \ldots k$, and assume that these parameters exhibit a weak type of decay:

$$\begin{aligned} \lambda_i/t_i &\geq CL, \quad i = 1, \ldots r-1, \\ \lambda_r/t_r &\leq \frac{(C+1)\theta^2 - 2}{2} L, \end{aligned} \tag{49}$$

for some $r < k$, with $r/k \to \theta \in (0,1)$, and some constant C. Then the limiting shrunken stagewise estimate $\tilde{x}(t)$ at the parameter value t, as $\epsilon \to 0$ and $\alpha \to 1$, satisfies

$$f(\tilde{x}(t)) = f(\hat{x}(t)),$$

i.e., $\tilde{x}(t)$ is a solution in (4) at the parameter value t.

Remark 1. The result above can be extended to the case when g is a seminorm. We simply need to redefine g^* and the updates in order to accomodate the possibly nontrivial null space N_g of g, as discussed in the third remark following Theorem 1.

Remark 2. The assumption in (49) of Theorem 2 stands out as technical assumption that is hard to interpret. This condition is used in the proof to control a term in the duality gap expansion that involves differences of $g^*(\nabla f(x^{(i)}))$ across successive iterations $i, i+1$. The theorem refers to such a quantity, $\lambda_i = g^*(\nabla f(x^{(i)}))$, as the "effective Lagrange parameter" at $x^{(i)}$. To explain this, consider the stationarity condition for the problem (4),

$$\nabla f(x) + \lambda v = 0,$$

where $v \in \partial g(x) = \mathrm{argmax}_{g^*(z) \leq 1} \; x^T z$. This implies that $\nabla f(x) = -\lambda v$, or $g^*(\nabla f(x)) = \lambda g^*(v) = \lambda$, which gives an expression for the Lagrange parameter associated with a solution of the constrained problem (4). As $x^{(i)}$ is not a solution, but an approximate one, we call $\lambda_i = g^*(\nabla f(x^{(i)}))$ its effective Lagrange parameter.

The condition (49) says that until some number of steps r along the path, the ratio of effective Lagrange parameters λ_i to bound parameters t_i must not be too small, and then at step r it must not be too large. This is a formulation of a type of weak decay of λ_i/t_i, $i = 1, 2, 3, \ldots$. It is not intuitively clear to us when (i.e., in what kinds of problems) we should expect this condition to be satisfied. We can, however, inspect it emprically. For the example lasso problem in Figure 9 (where, recall, the shrunken stagewise path appears to approach the exact solution path), we plot the ratio λ_i/t_i, $i = 1, 2, 3, \ldots$ in Figure 10. This ratio displays a sharp decay across steps of the algorithm, and so, at least empirically, the assumption (49) seems reasonable. We suspect that in general, the two hard bounds in (49) can be replaced by a more natural decay condition, and furthermore, there are characterizable problem classes with sharp decays of the Lagrange to bound parameter ratios. These are topics for future work.

6 Discussion

We presented a framework for computing incremental stagewise paths in a general regularized estimation setting, defined by minimizing a differentiable convex loss function subject to a convex constraint. The stagewise estimates are explicitly and efficiently computable for a wide variety of problems, and they provide an approximate solution path for the underlying convex problem of interest, but exhibit generally more stability as the regularization parameter changes. In some situations this approximation (i.e., the discrepancy between stagewise estimates and solutions) appears empirically to be quite tight, and in others it does not. All in all, however, we have found that the stagewise estimates essentially always offer competitive statistical performance (as measured, e.g.,

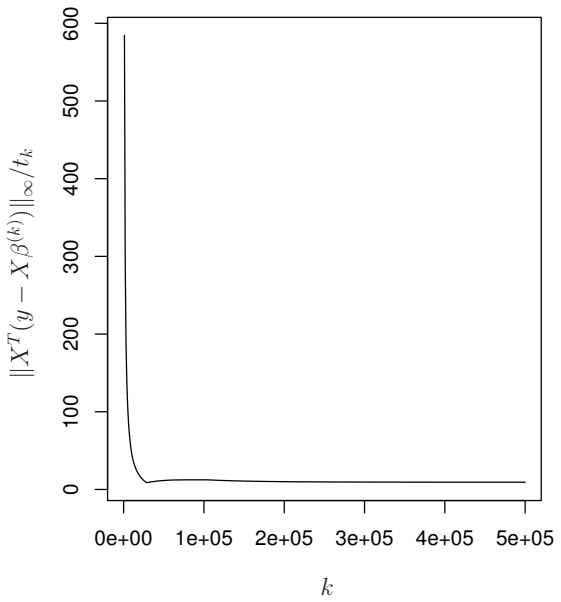


Figure 10: *A plot of* $\lambda_k/t_k = \|X^T(y - X\beta^{(k)})\|_\infty/t_k$ *across steps* k *of the shrunken stagewise algorithm, for the lasso data set of Figure 9. This decay roughly verifies the condition* (49) *of Theorem 2, needed to ensure the convergence of shrunken stagewise estimates to exact solutions.*

by test error) with that of exact solutions. This suggests that they should be a point of study, even apart from their ability to approximate solution paths of convex problems, and a rigorous (theoretical) characterization of the statistical properties of stagewise estimates is an important direction to pursue in the future. There are many other potential topics for future work, as alluded to throughout the paper. It is our hope that other researchers will take an interest too, and that this paper marks the beginning of a deeper understanding of stagewise capabilities.

Acknowledgements

This work was motivated by an attempt to explain the intuitive connection between forward stagewise regression and the lasso, in preparing lectures for a graduate class on optimization at Carnegie Mellon University. We thank co-teacher Geoff Gordon and the students of this class for early inspiring conversations. We also thank Rob Tibshirani, Jerry Friedman, Jonathan Taylor, Jacob Bien, and Lester Mackey for their helpful feedback. We are grateful to Jacob Bien for his understanding and patience throughout our (unusually slow) writing process, and to Lester Mackey for enlightening discussion on the Frank-Wolfe connection. Lastly, we would like to thank the editors and referees who reviewed this paper, as they provided extremely helpful and constructive reports. We gratefully acknowledge the funding support from NSF grant DMS-1309174.

A Appendix

A.1 Comparison to Frank-Wolfe

We compare our general stagewise procedure to the Frank-Wolfe algorithm for the general convex minimization problem (4). At any fixed value of t, the Frank-Wolfe algorithm begins with $\tilde{x}^{(0)} \in \mathbb{R}^n$ such that $g(\tilde{x}^{(0)}) \leq t$, and repeats the following steps (Frank & Wolfe 1956, Jaggi 2013):

$$\tilde{x}^{(k)} = (1-\gamma)\tilde{x}^{(k-1)} + \gamma\tilde{\Delta}, \tag{50}$$

$$\text{where } \tilde{\Delta} \in \operatorname*{argmin}_{z \in \mathbb{R}^n} \langle \nabla f(\tilde{x}^{(k-1)}), z \rangle \text{ subject to } g(z) \leq t, \tag{51}$$

$$\text{and } \gamma = 2/(k+1), \tag{52}$$

for $k = 1, 2, 3, \ldots$. The Frank-Wolfe steps can be seen as iteratively minimizing local linear approximations of the loss function f over the constraint set $\{x : g(x) \leq t\}$, as is done in (51). The actual updates performed in (50) take successively smaller and smaller steps in the direction of these local minimizers. Under fairly weak conditions, the Frank-Wolfe iterates satisfy $f(\tilde{x}^{(k)}) \to f(\hat{x}(t))$ as $k \to \infty$; in fact, as shown in, e.g., Jaggi (2013), the error $f(\tilde{x}^{(k)}) - f(\hat{x}(t))$ is $O(1/k)$. Jaggi (2013) also shows how to use the Frank-Wolfe iterates to easily compute a duality gap for the problem (4), so in practice we could stop iterating when this duality gap is sufficiently small.

At face value, the Frank-Wolfe steps (50), (51), (52) and the stagewise steps (5), (6) appear very similar. One apparent difference is that the former steps are iterated to ultimately yield a single estimate at a given value of the regularization parameter t, whereas the latter steps are iterated to yield several estimates (one per iteration) that form a regularization path. We make more substantial and informative comparisons between the two methods below.

First, consider a setting in which we run the Frank-Wolfe algorithm multiple times, in order to compute estimates at multiple values of the regularization parameter t; a typical strategy would be to run the Frank-Wolfe algorithm until convergence at each desired value of t, using "warm starts" (i.e., at the end of each run, we would use the newly computed estimate as the initial guess $\tilde{x}^{(0)}$ in the Frank-Wolfe algorithm at the next parameter value). With this in mind, it may be tempting to compare our stagewise algorithm to something like a 1-step Frank-Wolfe algorithm, where at each regularization parameter value, we perform a single Frank-Wolfe update to construct our estimate, rather than iterating the algorithm until convergence. But a more careful examination shows that these two approaches, the stagewise and 1-step Frank-Wolfe approaches, are actually quite different. To make the comparison as direct as possible, assume that the 1-step Frank-Wolfe procedure starts with $x^{(0)} = \hat{x}(t_0)$, a solution in (4) at $t = t_0$. It would then compute estimates $x^{(k)}$, $k = 1, 2, 3, \ldots$ at the regularization parameter values $t_k = t_{k-1} + \epsilon$, $k = 1, 2, 3, \ldots$ via

$$x^{(k)} = \tilde{\Delta}, \tag{53}$$

$$\text{where } \tilde{\Delta} \in \operatorname*{argmin}_{z \in \mathbb{R}^n} \langle \nabla f(x^{(k-1)}), z \rangle \text{ subject to } g(z) \leq t_k, \tag{54}$$

which is just a single step of the Frank-Wolfe algorithm at $t = t_k$, taking as the initial guess $x^{(k-1)}$. We can see that both the 1-step Frank-Wolfe (53), (54) and stagewise (5), (6) updates utilize a local linearization of f around previous estimate, and minimize this linear function over a sublevel set of g, but they do so in subtly different ways. The 1-step Frank-Wolfe approach takes $x^{(k)}$ to be a minimizer of $\langle \nabla f(x^{(k-1)}), z \rangle$ over the full constraint set $\{z : g(z) \leq t_k\}$; the stagewise approach computes a minimizer of $\langle \nabla f(x^{(k-1)}), z \rangle$ over a highly restricted constraint set $\{z : g(z) \leq \epsilon\}$, and adds this to the last estimate $x^{(k-1)}$ to form $x^{(k)}$. In both cases, the estimate $x^{(k)}$ is a feasible point for problem (4) at $t = t_k$. See Figure 11 for an illustration.

Though seemingly similar, these two strategies result in entirely different paths of estimates. Generally speaking, the 1-step Frank-Wolfe strategy (53), (54) is not very useful, since its update

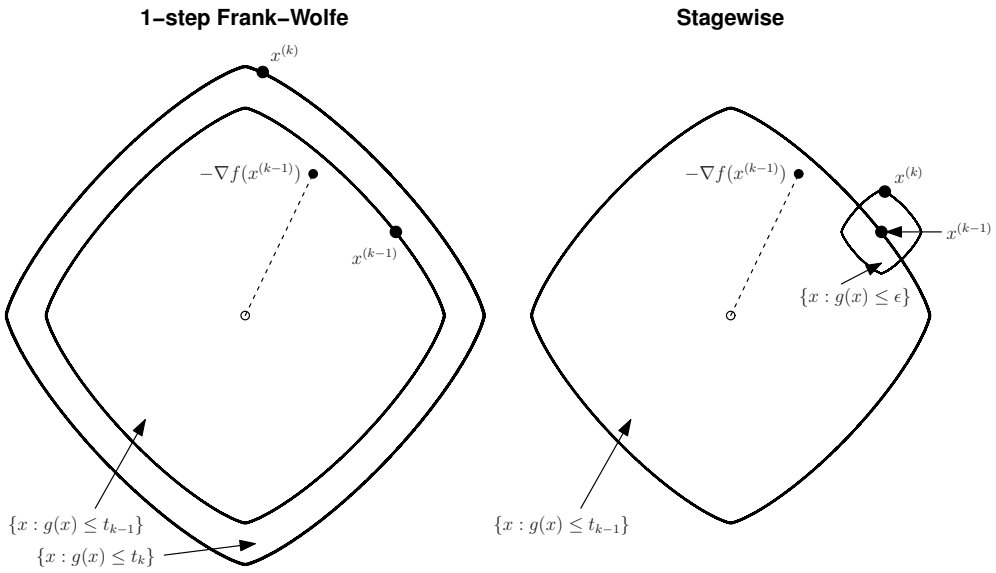


Figure 11: *Illustration of the 1-step Frank-Wolfe and stagewise methods. Each starts with an estimate $x^{(k-1)}$ at a regularization parameter value t_{k-1}, satisfying $g(x^{(k-1)}) \le t_{k-1}$, i.e., a feasible point for the problem* (4) *(but not necessarily optimal). At a larger parameter value $t_k = t_{k-1} + \epsilon$, the 1-step Frank-Wolfe strategy inflates the constraint set to $\{x : g(x) \le t_k\}$, and chooses its estimate $x^{(k)}$ to be the point most aligned with $-\nabla f(x^{(k-1)})$ over this new constraint set. Note that this means $x^{(k)}$ can be far away from the previous estimate $x^{(k-1)}$ at t_{k-1}. (Note also that the typical Frank-Wolfe strategy, as opposed to the 1-step strategy, would not settle for such a point $x^{(k)}$ as its estimate at t_k, but would continue iterating from $x^{(k)}$ by repeatedly minimizing linear approximations of f over $\{x : g(x) \le t_k\}$ until convergence.) The stagewise strategy instead builds a shrunken constraint set $\{x : g(x) \le \epsilon\}$ around $x^{(k-1)}$, and considers only the points in this small region as candidates for its next estimate. It then constructs $x^{(k)}$ using the same logic as above, by finding the point maximally aligned with $-\nabla f(x^{(k-1)})$ over the new constraint region. Such differences (between the Frank-Wolfe and stagewise strategies) may not seem drastic, but they have big implications.*

steps discard too much information from previous estimates. Consider, e.g., the ℓ_1 regularization setting, where $g(x) = \|x\|_1$: here each estimate from the 1-step Frank-Wolfe algorithm would have only one nonzero component, corresponding to the maximum absolute entry of the gradient vector evaluated at the previous estimate.[7] Hence, instead of producing a sequence of models that become progressively more and more dense as the regularization parameter increases, as with the stagewise algorithm, the 1-step Frank-Wolfe algorithm produces a sequence of trivial models, each with just one active variable. Similar conclusions can be drawn by looking at settings like group-structured regularization, trace norm regularization, etc.

We would likely never use the 1-step Frank-Wolfe procedure in practice to compute an (approximate) regularization path, but the insights gained from studying this algorithm carry over to the more common use case introduced initially: the typical Frank-Wolfe strategy, in which we run the Frank-Wolfe algorithm until convergence across a sequence of regularization parameter values t_k, $k = 1, 2, 3, \ldots$ with warm starts, discards a lot of information about previously computed estimates. At a parameter value t_k, the only information used by the Frank-Wolfe algorithm about the previously computed estimate $x^{(k-1)}$ is the gradient of f at $x^{(k-1)}$. In particular, in its first step at t_k,

[7]Strictly speaking, if there are ties between the absolute components of the gradient vector at t_k, then the estimate can be taken to be any convex combination $t_k \cdot \sum_{i \in \mathcal{I}} \alpha_i e_i$, where $\mathcal{I}$ is the set of maximizing indices, and each $\alpha_i \ge 0$ with $\sum_{i \in \mathcal{I}} \alpha_i = 1$. We do not maintain this distinction throughout our discussion in this section.

it chooses the first iterate to minimize the inner product with $\nabla f(x^{(k-1)})$ over all feasible points. If this minimizer is far from $x^{(k-1)}$, then, assuming that the solutions at t_{k-1} and t_k are close, the Frank-Wolfe algorithm basically wastes iterations bringing itself back to where it was at the end of its run for t_{k-1}. Interestingly, the iterations *within* a run of Frank-Wolfe at a fixed parameter value t_k prevent the algorithm from deviating too far from previous iterates, by means of the shrinkage factor γ in (52); however, no such control takes place *between* runs of the Frank-Wolfe algorithm at successive parameter values, t_{k-1} and t_k, using warm starts. The stagewise algorithm (5), (6), on the other hand, shares a great deal of information between estimates at successive iterations (recall that by definition, the stagewise estimates $x^{(k)}$ and $x^{(k-1)}$ differ by an amount Δ, controlled to be small under g), and in this sense, it makes a much more efficient use of its history.

It helps to think about an example. Returning to the ℓ_1 regularization setting, suppose that we have computed an estimate $x^{(k-1)}$ with, say, 50 nonzero components out of 1000 at some value of the regularization parameter t_{k-1}. At a slightly larger parameter value t_k, the stagewise algorithm retains essentially all of the information in $x^{(k-1)}$—information about which variables are active, and the values of their coefficients—and increments (or decrements) another component of $x^{(k-1)}$ in order to form $x^{(k)}$. By comparison, the Frank-Wolfe algorithm uses $x^{(k-1)}$ as a warm start for its run at t_k, meaning that for its first step, it constructs an iterate with only one nonzero component, corresponding to the maximal entry of $\nabla f(x^{(k-1)})$ in absolute value. In subsequent steps, only one component of the iterate is adjusted at a time. Said in words, the Frank-Wolfe algorithm at t_k has to "relearn" the entire set of active variables (and their coefficient values), starting from the empty set. This seems like a markedly inefficient use of its computational history, certainly in comparison to the strategy taken by the stagewise algorithm.[8]

Of course, a crucial difference to note is that the estimates $x^{(k)}$, $k = 1, 2, 3, \ldots$ from the Frank-Wolfe strategy are guaranteed to be solutions in (4) (up to an arbitrarily small level of tolerance) at $t = t_k$, $k = 1, 2, 3, \ldots$, but the stagewise estimates $x^{(k)}$, $k = 1, 2, 3, \ldots$ are not, even as the spacings between the parameter values t_k, $k = 1, 2, 3, \ldots$ goes to zero. One can also argue that the Frank-Wolfe algorithm was not designed to be a path following method, and so comparing to stagewise by simply applying it sequentially with warm starts is unfair. Some authors have in fact considered a specialized Frank-Wolfe strategy for path following (Giesen, Jaggi & Laue 2012*a*,*b*, Giesen, Laue, Muller & Swiercy 2012). The general goal of this work is to construct an approximate solution path in (4) with a provable approximation guarantee (in terms of the achieved criterion value); this is done by continuously controlling a duality gap for problem (4) as the parameter t varies, a strategy that does not depend on the Frank-Wolfe algorithm per se, but can be easily combined with the Frank-Wolfe algorithm because its iterates readily admit such a duality gap.

An implementation of this idea is described in Appendix A.2, as its details are not important for the current discussion. This path following algorithm can be setup to ensure a γ-suboptimal regularization path, for any given $\gamma > 0$, and operationally it boils down to running Frank-Wolfe with warm starts over a sequence of adaptively chosen parameter values t_k, $k = 1, 2, 3, \ldots$ (rather than a given fixed sequence). This adaptive sequence tends to be quite dense for reasonably small choices of γ (much more dense than a typical fixed sequence of parameter values); for larger values of γ, the adaptive sequence is more spread out, but then it takes many iterations at each parameter value to converge (especially towards the unregularized end of the path). Altogether, the previous comparison between the two methods can be drawn here: the Frank-Wolfe path following strategy does not utilize its history nearly as efficiently as the stagewise algorithm.

The arguments in this subsection were based on high level reasoning, but they are empirically supported by the examples in Section 4 and Appendix A.2, where we run stagewise and Frank-Wolfe

[8]For completeness, we should also mention a variant of the Frank-Wolfe algorithm proposed by Jaggi (2013), in which the shrinkage parameter γ in the update step (50) is chosen by exact line search, as opposed to the default (nonadaptive) value given in (52). This version of Frank-Wolfe has the potential to use more of its history, depending on how large it sets γ (especially in its first step). Still, a key distinction remains: the adaptive Frank-Wolfe strategy can choose to use more or less of its history, but for the stagewise algorithm, relying strongly on the computed history is a set decision, not one that is adaptively made over its course.

across a variety of scenarios. We can summarize the comparisons drawn, as follows:

- the Frank-Wolfe algorithm, run over a (fixed or adaptively chosen) sequence of regularization parameter values with warm starts, does not make an efficient use of its computational history (i.e., the information contained in previously computed estimates), however, it is guaranteed to produce solutions in (4);
- the stagewise algorithm is comparatively much more efficient at using its history of estimates, but is not guaranteed to produce solutions in (4).

The fact that the Frank-Wolfe algorithm relinquishes so much information about previously computed estimates may actually be the reason, roughly speaking, that it is able to produce solutions in (4). After all, the solution path of the convex regularization problem (4) can be highly variable (e.g., in a high-dimensional lasso problem with correlated predictors, the components of the solution path can be very wiggly, as predictors can enter and leave the active set many times), and therefore, by not constraining itself to adhere strongly to its computational past, the Frank-Wolfe algorithm gives itself the freedom to fit each individual estimate (along a sequence of parameter values t_k, $k = 1, 2, 3, \ldots$) as appropriate. In contrast, the stagewise algorithm is constrained to closely follow its path of previously computed estimates, by construction. One can even look at this constrained nature of fitting as an additional type of regularization. Except in special circumstances (e.g., monotone component paths in the lasso problem), the stagewise algorithm does not produce exact solutions in (4), a seemingly necessary feature of any estimation method that follows its computational history so carefully. But this is not the end of the story; recall that a main theme of this paper (the third point in Section 1.1) is that the stagewise estimates are statistically useful in their own right, in spite of their (sometimes extreme) differences to solutions in (4). If we view the momentum that the stagewise method places on past estimates as an added level of regularization, then such a claim is perhaps not too surprising.

A.2 Path following with Frank-Wolfe

Assume that g is a norm, and $g^*(x) = \max_{g(z) \leq 1} x^T z$ is its dual norm. We propose below a path following strategy to compute an approximate regularization path with Frank-Wolfe.

Algorithm 4 (**Path following with Frank-Wolfe**).

Fix $\gamma, m > 0$, and $t_0 \in \mathbb{R}$. Set $\tilde{x}(t_0) = \hat{x}(t_0)$, a solution in (4) at $t = t_0$. Repeat, for $k = 1, 2, 3, \ldots$:

- *Calculate*
$$t_k = t_{k-1} + \frac{(1 - 1/m)\gamma}{g^*\big(\nabla f\big(\tilde{x}(t_{k-1})\big)\big)},$$
and set $\tilde{x}(t) = \tilde{x}(t_{k-1})$ for all $t \in [t_{k-1}, t_k)$.
- *Use Frank-Wolfe to compute $\tilde{x}(t_k)$, an (approximate) solution in (4) at $t = t_k$, having duality gap at most γ/m.*

One might notice that the above algorithm differs somewhat from the path following algorithms in Giesen, Jaggi & Laue (2012*a*,*b*), Giesen, Laue, Muller & Swiercy (2012) (specifically, in the way that it handles the regularization parameter t in (4)); we make modifications that we feel simplify the path following algorithm in the current setting, but really the main idea follows entirely the work of these authors. Algorithm 4 constructs a piecewise constant regularization path, $\tilde{x}(t)$, $t \geq t_0$. It begins by computing a solution in (4) at some initial value $t = t_0$ (and at a higher level of accuracy than the standard set for the overall path), increases t until the computed solution no longer meets the standard of accuracy as measured by the duality gap, recomputes a solution at this new value of the parameter, increases t, and so on.

It is easy to verify that the path output by this algorithm is feasible for (4) at all visited values of the parameter t; moreover, the path has the approximation property

$$f(\tilde{x}(t)) - f(\hat{x}(t)) \leq \gamma, \quad \text{for all } t. \tag{55}$$

(By all t, in the above, we mean all values of $t \geq t_0$ visited by the path algorithm.) To show this, we begin by remarking, as in Jaggi (2013), that the quantity

$$h_t(x) = \max_{g(z) \leq t} \langle \nabla f(x), x - z \rangle,$$

serves as a valid duality gap for problem (4), in that for all feasible x,

$$f(x) - f(\hat{x}(t)) \leq h_t(x),$$

It will be helpful to use an equivalent representation of the duality gap:

$$h_t(x) = \langle \nabla f(x), x \rangle + t \cdot \max_{g(z) \leq 1} \langle \nabla f(x), z \rangle = \langle \nabla f(x), x \rangle + t \cdot g^*\big(\nabla f(x)\big), \tag{56}$$

where we have used the fact that $g(az) = |a| g(z)$, and the definition of the dual norm g^*.

Now the argument for (55) is straightforward. By construction, at step k, we compute $\tilde{x}(t_k)$ to be an approximate solution with the property that $h_{t_k}(\tilde{x}(t_k)) \leq \gamma/m$. Since the algorithm assigns $\tilde{x}(t) = \tilde{x}(t_k)$ for all $t \leq t_{k+1}$, we must check that $h_t(\tilde{x}(t_k)) \leq \gamma$ for all $t \leq t_{k+1}$. Directly from (56),

$$\begin{aligned} h_t(\tilde{x}(t_k)) &= \langle \nabla f(\tilde{x}(t_k)), \tilde{x}(t_k) \rangle + t \cdot g^*\big(\nabla f(\tilde{x}(t_k))\big) \\ &\leq \gamma/m + (t - t_k) \cdot g^*\big(\nabla f(\tilde{x}(t_k))\big). \end{aligned}$$

As $t_{k+1} - t_k = (\gamma - \gamma/m)/g^*(\nabla f(\tilde{x}(t_k)))$, the result follows.

We now report on an example of Frank-Wolfe path following in group lasso regression, with the data setup as in Figure 4 (in the case of uncorrelated predictors). We chose $\gamma = 250$, hand-tuned to be the largest possible value of the duality gap bound so that resulting Frank-Wolfe path estimates differed no more in mean squared error from the exact solutions than the stagewise ones did (with a step size $\epsilon = 1$). This was measured by the maximum discrepancy in mean squared error over the 100 regularization parameter values at which exact solutions were computed; linear interpolation was used to compute mean squared errors for Frank-Wolfe and stagewise at these parameter values. See the left panel in Figure 12 for mean squared error curves. Under this large value of γ, and $m = 5$ (the results did not really change by varying m), the path following strategy produced an adaptive sequence of only 102 regularization parameter values spanning the full path range. However, it took many iterations at each parameter value (beyond the start of the path) to meet the required duality gap, as shown in the right panel of Figure 12. The total number of iterations used by the Frank-Wolfe path following method was over 14,000, which is extremely inefficient, especially viewed next to the 250 iterations needed by stagewise. To emphasize: the comparison here is quite clear-cut, because iterations of stagewise and Frank-Wolfe are computationally equivalent, and the two methods have been tuned to yield the same mean squared error performance.

A.3 Small example: fused lasso signal approximation

For a small 1d fused lasso example, in the Gaussian signal approximator setup with $n = 20$, we generated a piecewise constant underlying sequence $\beta^* \in \mathbb{R}^{20}$ with 5 segments (the levels were drawn uniformly at random between 1 and 10), and we added $N(0, 1)$ noise to form the observations $y \in \mathbb{R}^{20}$. Figure 13a displays the 1d fused lasso solution path on the left, and the stagewise path on the right, constructed from 900 steps using $\epsilon = 0.01$. The two paths look basically the same. In a rough sense, this is not too surprising, because the 1d fused lasso problem can be rewritten as a lasso

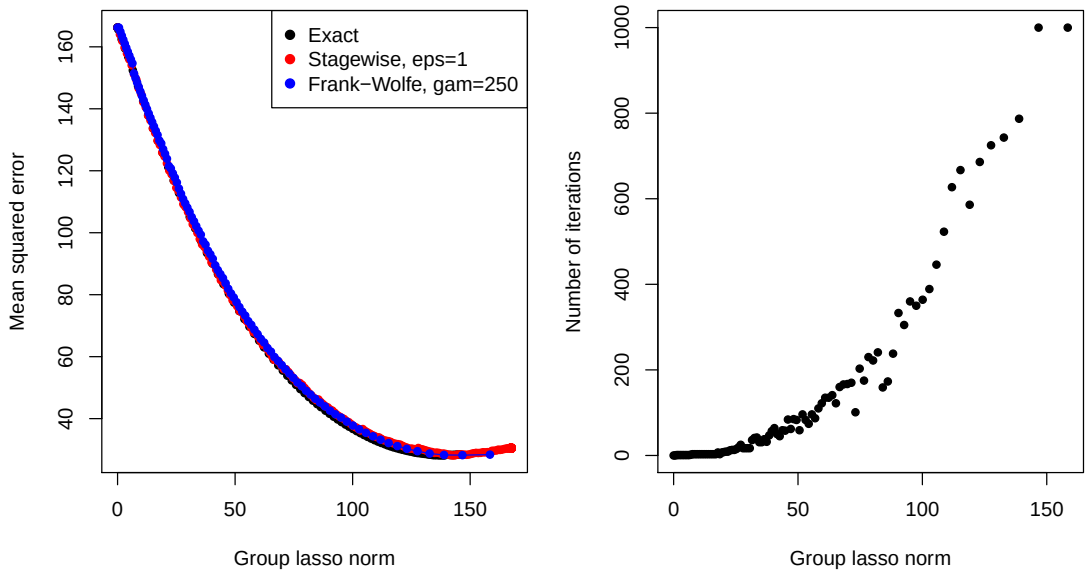
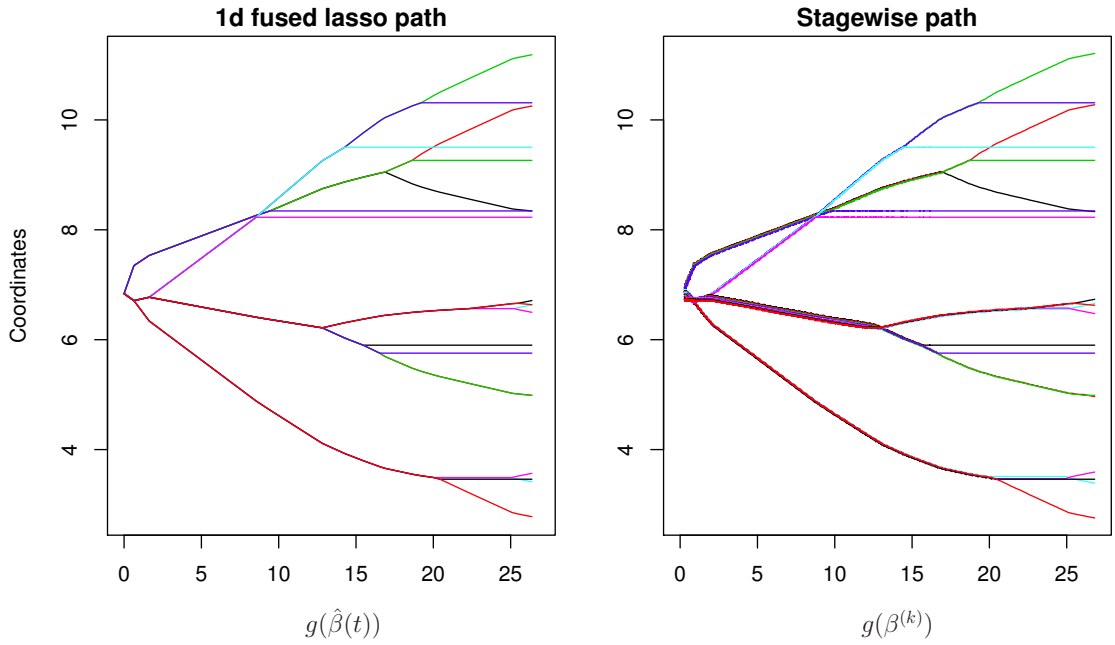


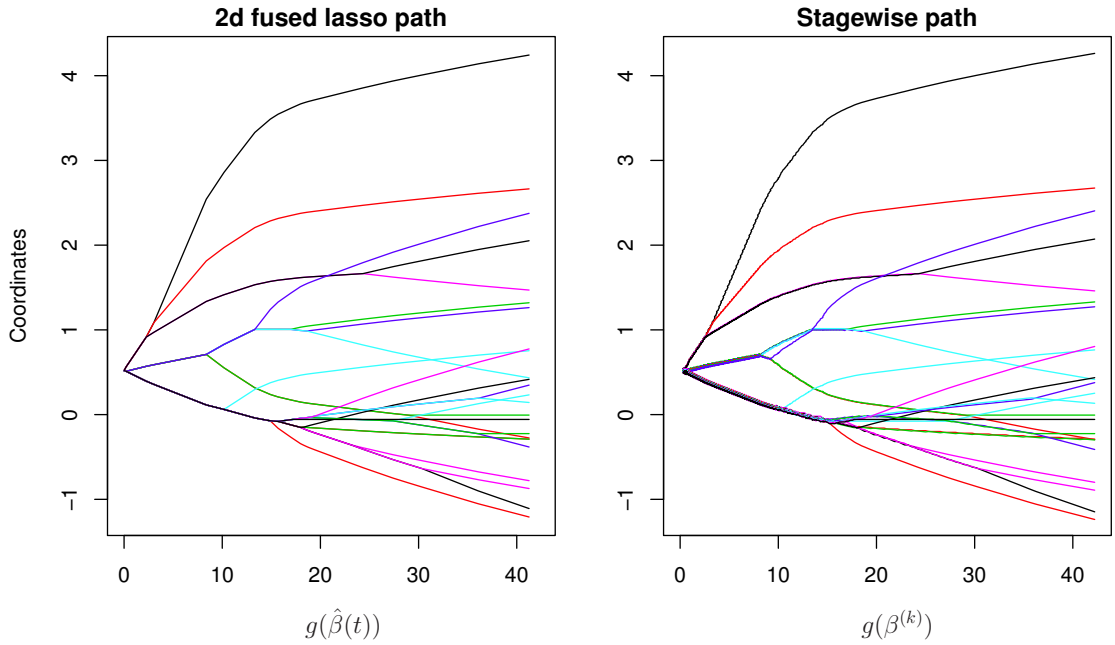
Figure 12: *For the same group lasso setup as in Figure 4 (in the uncorrelated predictors case), we ran the Frank-Wolfe path following strategy with* $\gamma = 250$ *and* $m = 5$*. The bound* γ *was chosen to be as large as possible so that the Frank-Wolfe estimates have competitive mean squared errors with the stagewise estimates and exact solutions, confirmed by the plot on the left. The right plot shows the number of iterations needed by Frank-Wolfe to converge at the required duality gap of* γ/m*, as a function of the visited regularization parameter value. The maximum number of iterations was 1000 (hence the algorithm did not converge for the largest two regularization parameter values.) We can see that a huge number of iterations are needed past the start of the path.*

problem with a predictor matrix X equal to the lower triangular matrix of 1s, and for this design, it is known that the limiting stagewise path (as $\epsilon \to 0$) is the exact solution path. (Here X satisfies the "positive cone condition", which ensures the lasso components paths are monotone, see Efron et al. (2004), Hastie et al. (2007).) But to be precise, this latter convergence result refers to the stagewise algorithm applied to the lasso parametrization directly, and this is not the same as applying the stagewise method to (the dual of) the 1d fused lasso parametrization, as was done in Figure 13a. It may be interesting to compare these two stagewise implementations, with the former iteratively adding step functions together, and the latter iteratively shrinking adjacent components towards each other. It may also be possible to prove a limiting equivalence between the latter stagewise method and the exact solution path, from arguments that rely on the monotonicity of the estimated differences, but we do not pursue these ideas in this paper.

For a 2d fused lasso example, still in the Gaussian signal approximator setup, we took $n = 25$, and $\beta^* \in \mathbb{R}^{25}$ to be an unraveled version of a piecewise constant 5×5 image. Pixels in the lower 2×2 corner of the image were assigned a common value of 3, and all other pixel values were zero. We formed $y \in \mathbb{R}^{25}$ by adding independent $N(0, 1)$ noise to β^*. The 2d fused lasso regularizer uses a 2-dimensional grid graph over the optimization variable $\beta \in \mathbb{R}^{25}$ i.e., this graph joins components of β that correspond to vertically or horizontally adjacent pixels in the image format. In Figure 13b, we show the exact 2d fused lasso solution path on the left, and the stagewise path on the right, from 500 steps with $\epsilon = 0.005$. It is not easy to spot many differences between the two (one difference can be seen when the 2d fused lasso norm is about 10), and altogether the stagewise path appears to be a very good approximation.



(a) *A small 1d fused lasso example with $n = 20$ points. The left plot shows the exact solution path, and the right plot shows the stagewise approximation, which is essentially identical.*



(b) *A small 2d fused lasso example, using a 5×5 image (so that $n = 25$). The solution path on the left and stagewise path on the right are only slightly different towards the regularized end of the paths.*

Figure 13: *Fused lasso examples in 1d and 2d.*

Finally, we emphasize that the stagewise steps in Figures 13a and 13b were derived from the dual, so the construction of paths proceeded *from right to left* in the stagewise plots (i.e., the stagewise paths were built for decreasing values of the regularization parameter t in (37)), contrary to all other stagewise examples outside of the generalized lasso setting. The stagewise approximation is hence most accurate at the right end of the plot, and its component paths become more choppy at the left end. In most foreseeable applications—fused lasso applications or otherwise—this is unfortunately not a desirable feature. Usually regularized estimates are of primary concern, so we would not want our iterative algorithm to reach these last, and certainly not with a lower measure of accuracy. An exception is the case of image denoising (under 2d fused lasso regularization): here estimates at low levels of regularization are usually interesting, as the underlying (noiseless) image itself is usually complex, especially for real, large images. The dual stagewise algorithm thrives in this case, as we saw on the large image denoising examples in Section 4.3.

A.4 Large example: ridge logistic regression

Overview. We investigate two simulated examples of ridge regularized logistic regression. The ridge logistic regression solutions were computed with the `glmnet` R package, available on CRAN, which offers a highly optimized coordinate descent implementation (Friedman et al. 2007, 2010). The `glmnet` package allows for lasso, ridge, and mixed (elastic net) regularization, and is actually more efficient in the presence of lasso regularization (because it utilizes an active set approach, which takes advantage of sparsity). However, it is still fairly efficient for pure ridge regularization if the number of variables p is not too large compared to the number of observations n, and the solutions to be computed correspond to large or moderate amounts of regularization (i.e., it does not need to compute solutions too close to the unregularized end of the path). Therefore we chose the example setups to meet these rough guidelines.

Recall, as described in Section 3.4, that the stagewise procedure for ridge regularization is very simple, and iteratively updates the estimate small amounts in the direction of the negative gradient (here, the gradient of the logistic loss function). Our C++ implementation of this stagewise routine, for the examples in the current section, is less than 25 lines of code. Meanwhile, the `glmnet` package uses a sophisticated, nuanced Fortran implementation of coordinate descent, which totals thousands of lines of code. (To be fair, the `glmnet` Fortran code is multipurpose, in that it solves more than just ridge regularized logistic regression: it handles elastic net regularized generalized linear models. Still, the broad comparison stands, between the complexities of the two implementations.)

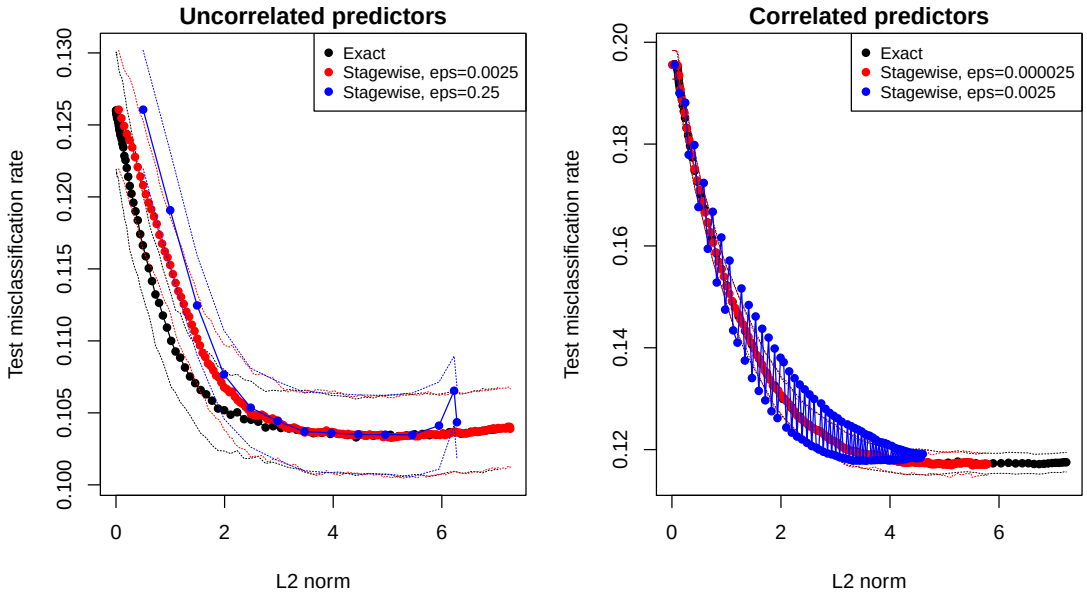
Examples and comparisons. Both simulation setups used $n = 8000$ observations and $p = 500$ predictor variables. The binary inputs $y \in \mathbb{R}^{8000}$ were drawn independently according to the logistic probabilities

$$p_i^* = \frac{1}{1 + \exp(-[X\beta^*]_i)}, \quad i = 1, \ldots 8000, \tag{57}$$

where the true coefficient vector $\beta^* \in \mathbb{R}^{500}$ had 50 nonzero components drawn independently from $N(0, 1)$, and the predictor matrix $X \in \mathbb{R}^{8000 \times 500}$ was constructed differently in the two setups. In the first, the entries of X were drawn independently from $N(0, 1)$, and in the second, the rows of X were drawn independently from $N(0, \Sigma)$, where $\Sigma \in \mathbb{R}^{500 \times 500}$ had unit diagonals and all off-diagonal elements equal to $\rho = 0.8$. In other words, the first setup used uncorrelated predictors and the second used highly positively correlated predictors.

In both cases, we ran `glmnet` over 100 regularization parameter values (starting from the regularized end of the path, using warm starts). We also ran the stagewise algorithm with two choices of step size, $\epsilon = 0.0025$ and $\epsilon = 0.25$. The results are shown in Figure 14. Looking at the uncorrelated case, in the left plot, first: we can see that, averaged over 10 simulated draws of the observations y (with fixed X, β^*), both stagewise sequences achieve a competitive minimum misclassification rate to that of the exact solution path (recorded with respect to independently drawn test inputs drawn

from (57)). In the early stages of the path, the exact solutions exhibit a better misclassification rate than the stagewise estimates with $\epsilon = 0.0025$, which in turn exhibit a better error rate than the stagewise estimates with $\epsilon = 0.25$, but all estimates end up at the same minimum misclassification rate later in the path. The table in the bottom row of Figure 14 shares the computation times for these methods (averaged over 10 draws of the observations, and recorded on a desktop computer). The `glmnet` coordinate descent implementation took an average of 12 seconds to compute its 100 solutions; stagewise with $\epsilon = 0.0025$ took about 3 seconds to compute 150 estimates; stagewise with $\epsilon = 0.25$ took 0.3 seconds to compute its 15 estimates.



Algorithm timings	
Method	Uncorrelated case
Exact: coordinate descent, 100 solutions	12.12 (0.12)
Stagewise: $\epsilon = 0.0025$, 150 estimates	3.01 (0.01)
Stagewise: $\epsilon = 0.25$, 15 estimates	0.30 (0.01)
Method	Correlated case
Exact: coordinate descent, 100 solutions	11.32 (0.31)
Stagewise: $\epsilon = 0.000025$, 2000 estimates	40.21 (0.24)
Stagewise: $\epsilon = 0.0025$, 1000 estimates	20.10 (0.18)

Figure 14: *Comparisons between exact and stagewise estimates for ridge regularized logistic regression, with $n = 8000$ and $p = 500$. The top two plots show test misclassification errors committed by solutions and stagewise estimates in two different scenarios, one with uncorrelated predictors on the left, and one with highly correlated predictors on the right. The bottom table gives timings for the* `glmnet` *coordinate descent algorithm in computing exact solutions and the stagewise algorithms. (All test errors and timings were averaged over 10 repetitions from the simulation model; the dotted lines in the plots show standard deviations, as do the parentheses in the table.) The stagewise algorithm performs ideally in the uncorrelated scenario, delivering statistically accurate estimates at very low computational cost; on the other hand, it seriously struggles in the correlated setup, requiring even smaller step sizes and far more steps to produce statistically meaningful estimates. (In the right plot, only 10% of the points along the stagewise error curves are drawn, and the standard deviations are withheld from the $\epsilon = 0.0025$ curve, for visibility.)*

In terms of the performance of the stagewise algorithm, the correlated problem setup stands in stark contrast to the uncorrelated one. In fact, this correlated case represents the closest incident to a failure for stagewise in this paper—to be perfectly clear, though, the "failure" here is entirely computational. Using step sizes $\epsilon = 0.000025$ and $\epsilon = 0.0025$, the stagewise method needed disproportionately more steps to cover a comparable part of the regularization path. This meant 2000 and 1000 steps when $\epsilon = 0.000025$ and $\epsilon = 0.0025$, respectively. Apparently the effective step length in this problem is greatly contracted, and the runtimes for computing a full stagewise regularization path are significantly inflated, as reported in the table in Figure 14. With the smaller step size, $\epsilon = 0.000025$, the right plot in Figure 14 shows that the stagewise estimates track the test misclassification rates of the exact solutions very closely; with the larger step size, $\epsilon = 0.0025$, the estimates display an odd trend in which their test errors bounce around the solution test errors.

This behavior, and the unusually slow stagewise progress, can be explained by the following rough geometric perspective. The contours of the logistic loss function $f(\beta)$ lie close to a tilted and very thin ellipse in $\mathbb{R}^{500}$, due to the highly positively correlated predictor variables X. (This contours are not exactly elliptical because the Hessian of f is not constant, but locally they are approximately so.) Starting from the origin, the stagewise algorithm repeatedly adds updates in the direction of the negative gradient of f. Because the ellipse is so thin, the updates will often pass "through" the ellipse and make little progress in advancing the ℓ_2 norm of the iterates. That is, the ℓ_2 norm of the iterates $\beta^{(k)}$, $k = 1, 2, 3, \ldots$ does not consistently increase, unless the step size ϵ is very small; otherwise progress it alternates back and forth, some steps advancing the ℓ_2 norm than others. Hence the stagewise coefficients plots, even with the fairly small step size $\epsilon = 0.0025$, display a distinct zigzag pattern; see Figure 15. This pattern is only exacerbated by multiplication by X, and the achieved error rates, which are based on the fitted values $X\beta^{(k)}$, $k = 1, 2, 3, \ldots$, jump around wildly. (It may be interesting to note that, despite this zigzag behavior, the mean squared error cruve between the stagewise estimates $\beta^{(k)}$ under $\epsilon = 0.0025$ and the true parameter β^* is actually still competitive, which is probably not surprising when staring at the strong similarities between coefficient paths in the top row of Figure 15.) With a small enough step size, $\epsilon = 0.000025$, this issue disappears, but of course the downside is that it now takes 2000 steps to compute a full stagewise path. A more thorough understanding of this problem and (hopefully) a computational remedy are important topics for future work.

A.5 Proof of Theorem 1

The suboptimality bound in this theorem is based on the quantity

$$h_t(x) = \max_{g(z) \leq t} \langle \nabla f(x), x - z \rangle.$$

As remarked in Appendix A.2, this serves as a valid duality gap for the problem (4), in that for all feasible x, we have $f(x) - f(\hat{x}(t)) \leq h_t(x)$, with $\hat{x}(t)$ being a solution in (4). This property follows directly from the first order condition for convexity applied to f. We note that if x is a solution in (4), then $h_t(x) = 0$, because in this case $\langle \nabla f(x), x \rangle \leq \langle \nabla f(x), z \rangle$ for all feasible z. Also, for the proof of the theorem, it will be helpful to rewrite $h_t(x)$ in the equivalent form:

$$h_t(x) = \langle \nabla f(x), x \rangle + t \cdot \max_{g(z) \leq 1} \langle \nabla f(x), z \rangle = \langle \nabla f(x), x \rangle + t \cdot g^*\big(\nabla f(x)\big),$$

as in Appendix A.2, relying on the fact that g is a norm, and g^* its dual norm. Lastly, it will be helpful to rewrite the stagewise updates (5), (6) as

$$x^{(k)} = x^{(k-1)} - \epsilon \delta^{(k-1)},$$
$$\text{where } \delta^{(k-1)} \in \operatorname*{argmax}_{z \in \mathbb{R}^n} \langle \nabla f(x^{(k-1)}), z \rangle \;\text{ subject to }\; g(z) \leq 1.$$

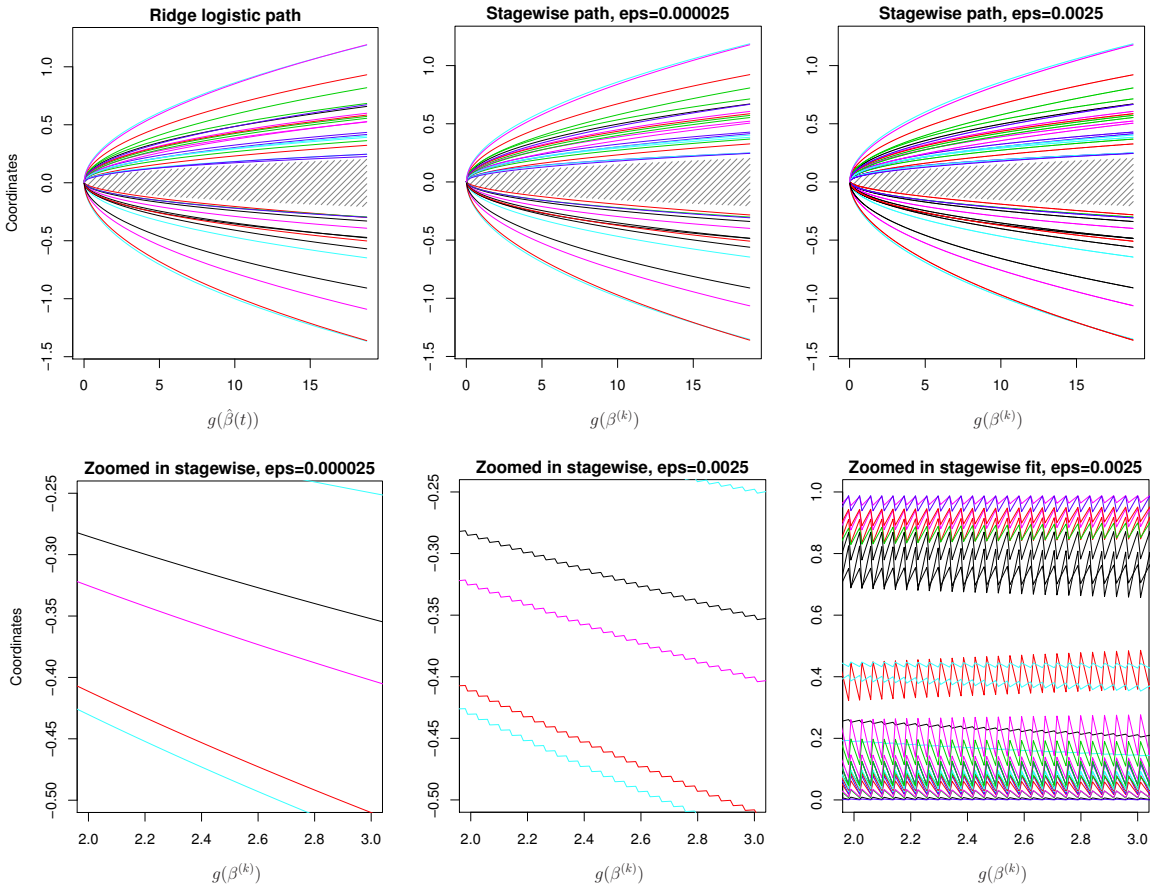


Figure 15: *Exact and stagewise coefficient plots for one simulated problem under the ridge regularized logistic regression setup, with correlated predictors. In the top row we excluded many of the coefficient paths that were close to zero (dashed gray region), and in the bottom row we zoomed in on a select number of coefficient paths, for visualization purposes.*

Proof of Theorem 1. At any arbitrary step k, we compute

$$\begin{aligned} h_{t_k}(x^{(k)}) &= \langle \nabla f(x^{(k)}), x^{(k)} \rangle + t_k g^*\big(\nabla f(x^{(k)})\big) \\ &= \langle \nabla f(x^{(k)}), x^{(k-1)} \rangle - \epsilon \langle \nabla f(x^{(k)}), \delta^{(k-1)} \rangle + t_k g^*\big(\nabla f(x^{(k)})\big). \end{aligned}$$

Now we add and subtract terms in order to express the right-hand side in terms of $h_{t_{k-1}}(x^{(k-1)})$,

$$\begin{aligned} h_{t_k}(x^{(k)}) &= \langle \nabla f(x^{(k-1)}), x^{(k-1)} \rangle + t_{k-1} g^*\big(\nabla f(x^{(k-1)})\big) + \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), x^{(k-1)} \rangle \\ &\quad - \epsilon \langle \nabla f(x^{(k)}), \delta^{(k-1)} \rangle + t_k g^*\big(\nabla f(x^{(k)})\big) - t_{k-1} g^*\big(\nabla f(x^{(k-1)})\big) \\ &= h_{t_{k-1}}(x^{(k-1)}) + \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), x^{(k-1)} \rangle - \epsilon \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), \delta^{(k-1)} \rangle \\ &\quad - \epsilon \langle \nabla f(x^{(k-1)}), \delta^{(k-1)} \rangle + t_k g^*\big(\nabla f(x^{(k)})\big) - t_{k-1} g^*\big(\nabla f(x^{(k-1)})\big) \\ &= h_{t_{k-1}}(x^{(k-1)}) + \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), x^{(k)} \rangle \\ &\quad - \epsilon \langle \nabla f(x^{(k-1)}), \delta^{(k-1)} \rangle + t_k g^*\big(\nabla f(x^{(k)})\big) - t_{k-1} g^*\big(\nabla f(x^{(k-1)})\big) \\ &= h_{t_{k-1}}(x^{(k-1)}) + \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), x^{(k)} \rangle + t_k \big[g^*\big(\nabla f(x^{(k)})\big) - g^*\big(\nabla f(x^{(k-1)})\big) \big]. \end{aligned}$$

Recursing this, we obtain

$$h_{t_k}(x^{(k)}) = \sum_{i=1}^{k} \langle \nabla f(x^{(i)}) - \nabla f(x^{(i-1)}), x^{(i)} \rangle + \sum_{i=1}^{k} t_i \big[g^*\big(\nabla f(x^{(i)})\big) - g^*\big(\nabla f(x^{(i-1)})\big) \big].$$

where we used the fact that $h_{t_0}(x^{(0)}) = 0$. For the first term, we can apply Hölder's inequality over the dual pair g, g^* to each summand; for the second term, we use the triange inequality $g^*(u - v) \geq g^*(u) - g^*(v)$. This yields

$$\begin{aligned}
h_{t_k}(x^{(k)}) &\leq \sum_{i=1}^{k} g^*\big(\nabla f(x^{(i)}) - \nabla f(x^{(i-1)})\big) g(x^{(i)}) + \sum_{i=1}^{k} t_i g^*\big(\nabla f(x^{(i)}) - \nabla f(x^{(i-1)})\big) \\
&\leq 2 \sum_{i=1}^{k} t_i g^*\big(\nabla f(x^{(i)}) - \nabla f(x^{(i-1)})\big) \\
&\leq 2L \sum_{i=1}^{k} t_i g(x^{(i)} - x^{(i-1)}) \\
&\leq 2L\epsilon \sum_{i=1}^{k} t_i.
\end{aligned}$$

In the second inequality, we used the fact that $g(x^{(i)}) \leq t_i$, and in the third, we invoked the Lipschitz assumption on ∇f. Since $t_i = t_0 + i\epsilon$, this upper bound is

$$h_{t_k}(x^{(k)}) \leq L\epsilon^2 k(k+1) + 2L\epsilon k t_0.$$

Finally, we recall that the number of steps k is chosen so that $t_k = t_0 + k\epsilon = t$, and hence

$$h_{t_k}(x^{(k)}) \leq L(t - t_0)^2 + L(t - t_0)\epsilon + 2L(t - t_0)t_0 = L(t^2 - t_0^2) + L(t - t_0)\epsilon,$$

which completes the proof. □

A.6 Proof of Theorem 2

This proof is similar to that of Theorem 1, although it is a little more involved technically. We will rely on a few limit calculations that are introduced and proved after the proof of the theorem, in Lemmas 6 and 7. We will also write the shrunken stagewise updates (47), (48) as

$$\begin{gathered}
x^{(k)} = \alpha x^{(k-1)} - \epsilon \delta^{(k-1)}, \\
\text{where } \delta^{(k-1)} \in \operatorname*{argmax}_{z \in \mathbb{R}^n} \langle \nabla f(x^{(k-1)}), z \rangle \;\text{ subject to }\; g(z) \leq 1.
\end{gathered}$$

Proof of Theorem 2. Assume without a loss of generality that $t_0 = 0$; the arguments needed for the case of an arbitrary starting value t_0 are similar but more tedious. As in the proof of Theorem 1, we compute the duality gap at an arbirary step k of the algorithm,

$$\begin{aligned}
h_{t_k}(x^{(k)}) &= \langle \nabla f(x^{(k)}), x^{(k)} \rangle + t_k g^*\big(\nabla f(x^{(k)})\big) \\
&= \alpha \langle \nabla f(x^{(k)}), x^{(k-1)} \rangle - \epsilon \langle \nabla f(x^{(k)}), \delta^{(k-1)} \rangle + t_k g^*\big(\nabla f(x^{(k)})\big), \\
&= \alpha \langle \nabla f(x^{(k-1)}), x^{(k-1)} \rangle + \alpha t_{k-1} g^*\big(\nabla f(x^{(k-1)})\big) + \alpha \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), x^{(k-1)} \rangle \\
&\quad - \epsilon \langle \nabla f(x^{(k)}), \delta^{(k-1)} \rangle + t_k g^*\big(\nabla f(x^{(k)})\big) - \alpha t_{k-1} g^*\big(\nabla f(x^{(k-1)})\big) \\
&= \alpha h_{t_{k-1}}(x^{(k-1)}) + \alpha \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), x^{(k-1)} \rangle - \epsilon \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), \delta^{(k-1)} \rangle
\end{aligned}$$

$$
\begin{aligned}
&\quad - \epsilon\langle \nabla f(x^{(k-1)}), \delta^{(k-1)}\rangle + t_k g^*\big(\nabla f(x^{(k)})\big) - \alpha t_{k-1} g^*\big(\nabla f(x^{(k-1)})\big) \\
&= \alpha h_{t_{k-1}}(x^{(k-1)}) + \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), x^{(k)}\rangle \\
&\quad - \epsilon\langle \nabla f(x^{(k-1)}), \delta^{(k-1)}\rangle + t_k g^*\big(\nabla f(x^{(k)})\big) - \alpha t_{k-1} g^*\big(\nabla f(x^{(k-1)})\big) \\
&= \alpha h_{t_{k-1}}(x^{(k-1)}) + \langle \nabla f(x^{(k)}) - \nabla f(x^{(k-1)}), x^{(k)}\rangle + t_k\big[g^*\big(\nabla f(x^{(k)})\big) - g^*\big(\nabla f(x^{(k-1)})\big)\big],
\end{aligned}
$$

and recursing this, we obtain

$$
h_{t_k}(x^{(k)}) = \underbrace{\sum_{i=1}^k \alpha^{k-i}\langle \nabla f(x^{(i)}) - \nabla f(x^{(i-1)}), x^{(i)}\rangle}_{A} + \underbrace{\sum_{i=1}^k \alpha^{k-i} t_i\big[g^*\big(\nabla f(x^{(i)})\big) - g^*\big(\nabla f(x^{(i-1)})\big)\big]}_{B}.
$$

We proceed to bound terms A and B separately.

Term A. We apply Hölder's inequality, and then use the Lipschitz continuity of ∇f,

$$
\begin{aligned}
A &\le \sum_{i=1}^k \alpha^{k-i} g^*\big(\nabla f(x^{(i)}) - \nabla f(x^{(i-1)})\big) g(x^{(i)}) \\
&\le L\sum_{i=1}^k \alpha^{k-i} t_i g(x^{(i)} - x^{(i-1)}) \\
&\le L\sum_{i=1}^k \alpha^{k-i} t_i\big((1-\alpha)t_{i-1} + \epsilon\big). \qquad (58)
\end{aligned}
$$

By definition, $t_i = \alpha t_{i-1} + \epsilon$ for all $i = 1, 2, 3, \ldots$, and a short inductive argument shows that

$$
t_i = \alpha^i t_0 + (\alpha^{i-1} + \ldots + \alpha + 1)\epsilon = \alpha^i t_0 + \frac{1-\alpha^i}{1-\alpha}\epsilon.
$$

Continuing from (58), we can rewrite the bound on term A as

$$
\begin{aligned}
A &\le \frac{L\epsilon^2}{1-\alpha}\sum_{i=1}^k \alpha^{k-i}(1-\alpha^i)(2-\alpha^{i-1}) \\
&= \frac{L\epsilon^2}{1-\alpha}\left(2\sum_{i=1}^k \alpha^{k-i} - 2\sum_{i=1}^k \alpha^k - \sum_{i=1}^k \alpha^{k-1} + \alpha^k \sum_{i=1}^k \alpha^{i-1}\right) \\
&= \frac{L\epsilon^2}{1-\alpha}\left(2\frac{1-\alpha^k}{1-\alpha} - 2\alpha^k k - \alpha^{k-1}k + \alpha^k \frac{1-\alpha^k}{1-\alpha}\right) \\
&\le \frac{L\epsilon^2}{(1-\alpha)^2}(1-\alpha^k)(2+\alpha^k) - \frac{3L\epsilon^2}{1-\alpha}\alpha^k k. \qquad (59)
\end{aligned}
$$

Let $M = \epsilon/((1-\alpha)t)$. Note that by the assumptions of the theorem, $M \to \infty$ as $\epsilon \to 0$ and $\alpha \to 1$. Hence, with a slight reparametrization, we will express the limits $\epsilon \to 0$, $\alpha \to 1$ as $\epsilon \to 0$, $M \to \infty$. Now we assume that the number of steps k is chosen so that

$$
t_k = \frac{1-\alpha^k}{1-\alpha}\epsilon = t,
$$

i.e., $1 - \alpha^k = 1/M$, or

$$
k = \frac{\log(1-1/M)}{\log\alpha} = \frac{\log(1-1/M)}{\log(1-\epsilon/(Mt))},
$$

where we have used that $\alpha = 1 - \epsilon/(Mt)$. Plugging in $\epsilon/(1-\alpha) = Mt$ and $1 - \alpha^k = 1/M$ into (59), our bound is

$$\begin{aligned} A &\leq LMt^2(3 - 1/M) - 3LMt(1 - 1/M)\epsilon k \\ &= \underbrace{3LtM(t - \epsilon k)}_{a} - Lt^2 + \underbrace{3Lt\epsilon k}_{b}. \end{aligned}$$

By Lemma 6, we have $a \to -3Lt^2/2$ as $\epsilon \to 0$, $M \to \infty$, and $b \to 3Lt^2$ as $\epsilon \to 0$, $M \to \infty$. Therefore, in the limit, we have

$$A \leq -3Lt^2/2 - Lt^2 + 3Lt^2 = Lt^2/2.$$

Term B. We decompose

$$B = \underbrace{\sum_{i=r+1}^{k} \alpha^{k-i} t_i \big[g^*\big(\nabla f(x^{(i)})\big) - g^*\big(\nabla f(x^{(i-1)})\big)\big]}_{c} + \underbrace{\sum_{i=1}^{r} \alpha^{k-i} t_i \big[g^*\big(\nabla f(x^{(i)})\big) - g^*\big(\nabla f(x^{(i-1)})\big)\big]}_{d},$$

and now consider each of c, d in turn.

Term c. Using the triangle inequality and the Lipschitz continuity of ∇f,

$$\begin{aligned} c &\leq \sum_{i=r+1}^{k} \alpha^{k-i} t_i g^*\big(\nabla f(x^{(i)}) - \nabla f(x^{(i-1)})\big) \\ &\leq L \sum_{i=r+1}^{k} \alpha^{k-i} t_i g(x^{(i)} - x^{(i-1)}) \\ &= \frac{L\epsilon^2}{1-\alpha} \sum_{i=r+1}^{k} \alpha^{k-i}(1-\alpha^i)(2-\alpha^{i-1}) \\ &= \underbrace{\frac{L\epsilon^2}{1-\alpha} \sum_{i=1}^{k} \alpha^{k-i}(1-\alpha^i)(2-\alpha^{i-1})}_{c_1} - \underbrace{\frac{L\epsilon^2}{1-\alpha} \sum_{i=1}^{r} \alpha^{k-i}(1-\alpha^i)(2-\alpha^{i-1})}_{c_2}. \end{aligned}$$

From the previous set of arguments, $c_1 \leq Lt^2/2$ as $\epsilon \to 0$, $M \to \infty$. Further, following these same arguments (but with r in place of k),

$$\begin{aligned} c_2 &= \alpha^{k-r} \frac{L\epsilon^2}{1-\alpha} \left(2\frac{1-\alpha^r}{1-\alpha} - 2\alpha^r r - \alpha^{r-1} r + \alpha^r \frac{1-\alpha^r}{1-\alpha} \right) \\ &\geq \alpha^{k-r} \left(\frac{L\epsilon^2}{(1-\alpha)^2}(1-\alpha^r)(2+\alpha^r) - \frac{3L\epsilon^2}{1-\alpha}\alpha^{r-1} r \right) \\ &= \alpha^{k-r} \left(\frac{L\epsilon^2}{(1-\alpha)^2}(1-\alpha^{k\frac{r}{k}})(2+\alpha^{k\frac{r}{k}}) - \frac{3L\epsilon^2}{1-\alpha}\frac{1}{\alpha}\alpha^{k\frac{r}{k}} k\frac{r}{k} \right) \\ &= \alpha^{k-r} \left(LM^2t^2\big(1-(1-1/M)^{\frac{r}{k}}\big)\big(2+(1-1/M)^{\frac{r}{k}}\big) - \frac{3LMt}{\alpha}(1-1/M)^{\frac{r}{k}} \epsilon k \frac{r}{k} \right) \\ &= \alpha^{k-r} \left(LM^2t^2\big(1-(1-1/M)^{\varphi}\big)\big(2+(1-1/M)^{\varphi}\big) - \frac{3LMt}{\alpha}(1-1/M)^{\varphi} \epsilon k \varphi \right). \end{aligned}$$

In the last line above, we have abbreviated $\varphi = r/k$, and we recall that $\varphi \to \theta$ as $\epsilon \to 0$, $M \to \infty$ by assumption. We expand the last expression, and omit the leading term α^{k-r} since it converges to 1:

$$\begin{aligned}
&LM^2t^2\big(1-(1-1/M)^{\varphi}\big)\big(2+(1-1/M)^{\varphi}\big) - \frac{3LMt}{\alpha}(1-1/M)^{\varphi}\epsilon k\varphi \\
&= 3LM^2t^2\big(1-(1-1/M)^{\varphi}\big) - \frac{3LMt}{\alpha}(1-1/M)^{\varphi}\epsilon k\varphi - LM^2t^2\big(1-(1-1/M)^{\varphi}\big)^2 \\
&= 3\varphi LtM(t-\epsilon k) + 3Lt^2M\Big(M\big(1-(1-1/M)^{\varphi}\big)-\varphi\Big) + \frac{3\varphi Lt\epsilon k}{\alpha}M\big(1-(1-1/M)^{\varphi}\big) \\
&\qquad - Lt^2M^2\big(1-(1-1/M)^{\varphi}\big)^2 + 3\varphi LtM\epsilon k(1/\alpha - 1) \\
&\to -3\theta Lt^2/2 + 3\theta(1-\theta)Lt^2/2 + 3\theta^2Lt^2 - \theta^2Lt^2 + 0 \\
&= \theta^2Lt^2/2.
\end{aligned}$$

The limits of the first four terms on the second to last line are due to Lemmas 6 and 7; the last term converges to 0 as $M(1-\alpha) = \epsilon/t \to 0$. Hence, in the limit, we have $c_2 \geq \theta^2Lt^2/2$, and

$$c \leq c_1 - c_2 \leq Lt^2/2 - \theta^2Lt^2/2 = (1-\theta^2)Lt^2/2.$$

Term d. We expand this term as

$$\begin{aligned}
d &= \sum_{i=1}^{r-1}\alpha^{k-1-i}(\alpha t_i - t_{i+1})g^*\big(\nabla f(x^{(i)})\big) + \alpha^{k-r}t_r g^*\big(\nabla f(x^{(r)})\big) - \alpha^{k-1}t_1 g^*\big(\nabla f(x^{(0)})\big) \\
&= -\epsilon\sum_{i=1}^{r-1}\alpha^{k-1-i}g^*\big(\nabla f(x^{(i)})\big) + \alpha^{k-r}t_r g^*\big(\nabla f(x^{(r)})\big) - \alpha^{k-1}t_1 g^*\big(\nabla f(x^{(0)})\big).
\end{aligned}$$

We now apply the bounds from the decay condition (49) in the theorem, which gives

$$\begin{aligned}
d &\leq -CL\epsilon\sum_{i=1}^{r-1}\alpha^{k-1-i}t_i + \frac{(C+1)\theta^2-2}{2}L\alpha^{k-r}t_r^2 \\
&= -\frac{CL\epsilon^2}{1-\alpha}\sum_{i=1}^{r-1}\alpha^{k-1-i}(1-\alpha^i) + \frac{(C+1)\theta^2-2}{2}L\alpha^{k-r}t_r^2 \\
&= \frac{-CL\epsilon^2}{(1-\alpha)^2}\alpha^{k-r}(1-\alpha^{r-1}) + \frac{CL\epsilon^2}{1-\alpha}\alpha^{k-1}(r-1) + \frac{(C+1)\theta^2-2}{2}L\alpha^{k-r}t_r^2 \\
&\leq \frac{-CL\epsilon^2}{(1-\alpha)^2}(\alpha^{k-r}-\alpha^{k-1}) + \frac{CL\epsilon^2}{1-\alpha}\alpha^{k-1}r + \frac{(C+1)\theta^2-2}{2}Lt^2 \\
&= -CLM^2t^2\Big((1-1/M)^{1-\varphi} - \frac{1}{\alpha}(1-1/M)\Big) + \frac{CLMt}{\alpha}(1-1/M)\epsilon k\varphi + \frac{(C+1)\theta^2-2}{2}Lt^2.
\end{aligned}$$

In the above, we have again written $\varphi = r/k$. Continuing with the first part of this upper bound,

$$\begin{aligned}
&- CLM^2t^2\Big((1-1/M)^{1-\varphi} - \frac{1}{\alpha}(1-1/M)\Big) + \frac{CLMt}{\alpha}(1-1/M)\epsilon k\varphi \\
&= CLM^2t^2\big(1/\alpha - (1-1/M)^{1-\varphi}\big) - \frac{CLMt^2}{\alpha}(1-\varphi) + \frac{\varphi CLt}{\alpha}M(\epsilon k - t) - \frac{\varphi CLt}{\alpha}\epsilon k \\
&= \frac{CLt^2}{\alpha}M\Big(M\big(1-(1-1/M)^{1-\varphi}\big) - (1-\varphi)\Big) + \frac{\varphi CLt}{\alpha}M(\epsilon k - t) - \frac{\varphi CLt}{\alpha}\epsilon k \\
&\qquad + CLt^2(1/\alpha - 1)M^2(1-1/M)^{1-\varphi}
\end{aligned}$$

$$\begin{aligned}
&\to \theta(1-\theta)CLt^2/2 + \theta CLt^2/2 - \theta CLt^2 + 0 \\
&= -\theta^2 CLt^2/2.
\end{aligned}$$

In the second to last line, the first three terms converge according to Lemmas 6 and 7. The last one converges to 0 since $M^2(1-\alpha) = \epsilon^2/((1-\alpha)t^2) \to 0$ by assumption. Therefore, in the limit, we have

$$d \le -\theta^2 CLt^2/2 + ((C+1)\theta^2 - 2)Lt^2/2 = (\theta^2 - 2)Lt^2/2.$$

Putting it all together. To finish, we have shown that in the limit as $\epsilon \to 0$, $M \to \infty$,

$$h_t(\tilde{x}(t)) = A + B \le Lt^2/2 + (1-\theta^2)Lt^2/2 + (\theta^2 - 2)Lt^2/2 = 0.$$

This completes the proof. □

Below are two helper lemmas used in the proof of Theorem 2.

Lemma 6. *For any fixed t, the following limits hold as $\epsilon \to 0$, $M \to \infty$:*

$$\begin{aligned}
\epsilon \frac{\log(1-1/M)}{\log(1-\epsilon/(Mt))} &\to t, \\
M\left(\epsilon \frac{\log(1-1/M)}{\log(1-\epsilon/(Mt))} - t\right) &\to t/2.
\end{aligned}$$

Proof. Both limits can be verified using a Taylor expansion of $\log(1+x)$ around $x = 0$. For the first, note that

$$\begin{aligned}
\epsilon \frac{\log(1-1/M)}{\log(1-\epsilon/(Mt))} &= \frac{\epsilon/M + \epsilon/(2M^2) + \epsilon/(3M^3) + \dots}{\epsilon/(Mt) + \epsilon^2/(2M^2t^2) + \epsilon^3/(3M^3t^3) + \dots} \\
&= \frac{1 + 1/(2M) + 1/(3M^2) + \dots}{1/t + \epsilon/(2Mt^2) + \epsilon^2/(3M^2t^3) + \dots} \\
&\to t.
\end{aligned}$$

And for the second,

$$\begin{aligned}
M\left(\epsilon \frac{\log(1-1/M)}{\log(1-\epsilon/(Mt))} - t\right) &= M\frac{1 + 1/(2M) + 1/(3M^2) + \dots - t\big(1/t + \epsilon/(2Mt^2) + \dots\big)}{1/t + \epsilon/(2Mt^2) + \epsilon^2/(3M^2t^3) + \dots} \\
&= \frac{1/2 + 1/(3M) + \dots - \big(\epsilon/(2t) + \epsilon^2/(3Mt^2) + \dots\big)}{1/t + \epsilon/(2Mt^2) + \epsilon^2/(3M^2t^3) + \dots} \\
&\to t/2.
\end{aligned}$$

□

Lemma 7. *Let $\varphi = \varphi(M)$ be such that $\varphi \to \theta$ as $M \to \infty$. The following limits hold, as $M \to \infty$:*

$$\begin{aligned}
M\big(1 - (1-1/M)^{\varphi}\big) &\to \theta, \\
M\Big(M\big(1 - (1-1/M)^{\varphi}\big) - \varphi\Big) &\to \theta(1-\theta)/2.
\end{aligned}$$

Proof. We use a Taylor expansion of $(1+x)^a$ around $x = 0$. In particular,

$$M\big(1 - (1-1/M)^{\varphi}\big) = M\left[\binom{\varphi}{1}\frac{1}{M} - \binom{\varphi}{2}\frac{1}{M^2} + \binom{\varphi}{3}\frac{1}{M^3} - \dots\right]$$

$$= \binom{\varphi}{1} - \binom{\varphi}{2}\frac{1}{M} + \binom{\varphi}{3}\frac{1}{M^2} - \dots$$
$$\to \binom{\theta}{1} = \theta.$$

(Note that here $\binom{x}{i}$ denotes the generalized binomial coefficient, for nonintegral x.) Also,

$$\begin{aligned} M\Big(M\big(1-(1-1/M)^{\varphi}\big) - \varphi\Big) &= M\left[\binom{\varphi}{1} - \binom{\varphi}{2}\frac{1}{M} + \binom{\varphi}{3}\frac{1}{M^2} - \dots - \varphi\right] \\ &= -\binom{\varphi}{2} + \binom{\varphi}{3}\frac{1}{M} - \dots \\ &\to -\binom{\theta}{2} = \frac{\theta(1-\theta)}{2}. \end{aligned}$$

□

References

Argyriou, A., Evgeniou, T. & Pontil, M. (2006), 'Multi-task feature learning', *Advances in Neural Information Processing Systems* **19**.

Bakin, S. (1999), Adaptive regression and model selection in data mining problems, PhD thesis, School of Mathematical Sciences, Australian National University.

Boyd, S. & Vandenberghe, L. (2004), *Convex Optimization*, Cambridge University Press, Cambridge.

Buhlmann, P. & Yu, B. (2010), 'Boosting', *Wiley Interdisciplinary Reviews: Computational Statistics* **2**(1), 69–74.

Candes, E. J. & Recht, B. (2009), 'Exact matrix completion via convex optimization', *Foundations of Computational Mathematics* **9**(6), 717–772.

Candes, E. J. & Tao, T. (2010), 'The power of convex relaxation: near-optimal matrix completion', *IEEE Transactions on Information Theory* **56**(5), 2053–2080.

Chambolle, A. & Darbon, J. (2009), 'On total variation minimization and surface evolution using parametric maximum flows', *International Journal of Computer Vision* **84**, 288–307.

Chen, J. & Ye, J. (2014), 'Sparse trace norm regularization', *Computational Statistics* **29**(3–4), 623–629.

Draper, N. & Smith, H. (1966), *Applied Regression Analysis*, Wiley, New York.

Efron, B., Hastie, T., Johnstone, I. & Tibshirani, R. (2004), 'Least angle regression', *Annals of Statistics* **32**(2), 407–499.

Efroymson, M. (1966), 'Stepwise regression—a backward and forward look', *Eastern Regional Meetings of the Institute of Mathematical Statistics* .

Eilers, P. & Marx, B. (1996), 'Flexible smoothing with B-splines and penalties', *Statistical Science* **11**(2), 89–121.

Frank, M. & Wolfe, P. (1956), 'An algorithm for quadratic programming', *Naval Research Logistics Quarterly* **32**(1–2), 95–110.

Friedman, J. (2001), 'Greedy function approximation: a gradient boosting machine', *Annals of Statistics* **29**(5), 1190–1232.

Friedman, J. (2008), Fast sparse regression and classification. Unpublished manuscript, `http://www-stat.stanford.edu/~jhf/ftp/GPSpub.pdf`.

Friedman, J., Hastie, T., Hoefling, H. & Tibshirani, R. (2007), 'Pathwise coordinate optimization', *Annals of Applied Statistics* **1**(2), 302–332.

Friedman, J., Hastie, T. & Tibshirani, R. (2010), 'Regularization paths for generalized linear models via coordinate descent', *Journal of Statistical Software* **33**(1), 1–22.

Friedman, J. & Popescu, B. (2004), Gradient directed regularization. Unpublished manuscript, `http://www-stat.stanford.edu/~jhf/ftp/pathlite.pdf`.

Giesen, J., Jaggi, M. & Laue, S. (2012*a*), 'Approximating parametrized convex optimization problems', *ACM Transactions on Algorithms* **9**(1), 1–17.

Giesen, J., Jaggi, M. & Laue, S. (2012*b*), 'Regularization paths with guarantees for convex semidefinite optimization problems', *Proceedings of the International Conference on Artificial Intelligence and Statistics* **15**.

Giesen, J., Laue, S., Muller, J. & Swiercy, S. (2012), 'Approximating concavely parametrized optimization problems', *Advances in Neural Information Processing Systems* **25**.

Golub, G. H. & Van Loan, C. F. (1996), *Matrix computations*, The Johns Hopkins University Press, Baltimore. Third edition.

Green, P. & Silverman, B. (1994), *Nonparametric Regression and Generalized Linear Models: A Roughness Penalty Approach*, Chapman & Hall/CRC Press, Boca Raton.

Harchaoui, Z., Douze, M., Paulin, M., Dudik, M. & Malick, J. (2012), 'Large-scale image classification with trace-norm regularization', *IEEE Conference on Computer Vision and Pattern Recognition* pp. 3386–3393.

Hastie, T., Taylor, J., Tibshirani, R. & Walther, G. (2007), 'Forward stagewise regression and the monotone lasso', *Electronic Journal of Statistics* **1**, 1–29.

Hastie, T., Tibshirani, R. & Friedman, J. (2009), *The Elements of Statistical Learning; Data Mining, Inference and Prediction*, Springer, New York. Second edition.

Hiriart-Urruty, J.-B. & Lemarechal, C. (1993), *Convex Analysis and Minimization Algorithms*, Springer, Berlin. Two volumes.

Hoerl, A. & Kennard, R. (1970), 'Ridge regression: biased estimation for nonorthogonal problems', *Technometrics* **12**(1), 55–67.

Jaggi, M. (2013), 'Revisiting frank-wolfe: Projection-free sparse convex optimization', *Proceedings of the International Conference on Machine Learning* **30**.

Jaggi, M. & Sulovsky, M. (2010), 'A simple algorithm for nuclear norm regularized problems', *Proceedings of the International Conference on Machine Learning* **27**.

Kelley, J. E. (1960), 'The cutting-plane method for solving convex programs', *Journal of the Society for Industrial and Applied Mathematics* **8**(4), 703–712.

Kim, S.-J., Koh, K., Boyd, S. & Gorinevsky, D. (2009), 'ℓ_1 trend filtering', *SIAM Review* **51**(2), 339–360.

Knudsen, E. (2013), Stagewise regression: competing with the state of the art via an efficient, simple, iterative algorithm, Undergraduate honors thesis, Department of Statistics, Carnegie Mellon University.

Mazumder, R., Hastie, T. & Tibshirani, R. (2010), 'Spectral regularization algorithms for learning large incomplete matrices', *Journal of Machine Learning Research* **11**, 2287–2322.

Meier, L., van de Geer, S. & Buhlmann, P. (2008), 'The group lasso for logistic regression', *Journal of the Royal Statistical Society: Series B* **70**(1), 53–71.

Obozinski, G., Taskar, B. & Jordan, M. (2010), 'Joint covariate selection and joint subspace selection for multiple classification problems', *Statistics and Computing* **20**(2), 231–252.

Ramsay, J. (2005), Parameter flows. Unpublished manuscript.

Rosset, S., Zhu, J. & Hastie, T. (2004), 'Boosting as a regularized path to a maximum margin classifier', *Journal of Machine Learning Research* **5**, 941–973.

Rudin, L. I., Osher, S. & Faterni, E. (1992), 'Nonlinear total variation based noise removal algorithms', *Physica D: Nonlinear Phenomena* **60**, 259–268.

Simon, N., Friedman, J., Hastie, T. & Tibshirani, R. (2013), 'A sparse group lasso', *Journal of Computational and Graphical Statistics* **22**(2).

Teo, C. H., Le, Q., Smola, A. & Vishwanathan, S. V. N. (2007), 'A scalable modular convex solver for regularized risk minimization', *Proceedings of the International Conference on Knowledge Discovery and Data Mining* **13**.

Teo, C. H., Vishwanathan, S. V. N., Smola, A. & Le, Q. (2010), 'Bundle methods for regularized risk minimization', *Journal of Machine Learning Research* **11**, 311–365.

Tibshirani, R. (1996), 'Regression shrinkage and selection via the lasso', *Journal of the Royal Statistical Society: Series B* **58**(1), 267–288.

Tibshirani, R. J. (2013), 'The lasso problem and uniqueness', *Electronic Journal of Statistics* **7**, 1456–1490.

Tibshirani, R. J. (2014), 'Adaptive piecewise polynomial estimation via trend filtering', *Annals of Statistics* **42**(1), 285–323.

Tibshirani, R. J. & Taylor, J. (2011), 'The solution path of the generalized lasso', *Annals of Statistics* **39**(3), 1335–1371.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J. & Knight, K. (2005), 'Sparsity and smoothness via the fused lasso', *Journal of the Royal Statistical Society: Series B* **67**(1), 91–108.

Tikhonov, A. (1943), 'On the stability of inverse problems', *Doklady Akademii Nauk SSSR* **39**(5), 195–198.

Turlach, B., Venables, W. & Wright, S. (2005), 'Simultaneous variable selection', *Technometrics* **47**(3), 349–363.

Wahba, G. (1990), *Spline Models for Observational Data*, Society for Industrial and Applied Mathematics, Philadelphia.

Wang, Y.-X., Sharpnack, J., Smola, A. & Tibshirani, R. J. (2015), 'Trend filtering on graphs', *Proceedings of the International Conference on Artificial Intelligence and Statistics* **18**, 1042–1050.

Yuan, M. & Lin, Y. (2006), 'Model selection and estimation in regression with grouped variables', *Journal of the Royal Statistical Society: Series B* **68**(1), 49–67.

Zhao, P. & Yu, B. (2007), 'Stagewise lasso', *Journal of Machine Learning Research* **8**, 2701–2726.