penalized: A MATLAB Toolbox for Fitting Generalized Linear Models with Penalties.

William McIlhagga
University of Bradford

Abstract

penalized is a flexible, extensible, and efficient MATLAB toolbox for penalized maximum likelihood. **penalized** allows you to fit a generalized linear model (gaussian, logistic, poisson, or multinomial) using any of ten provided penalties, or none. The toolbox can be extended by creating new maximum likelihood models or new penalties. The toolbox also includes routines for cross-validation and plotting.

Keywords: generalized linear models, penalized regression, lasso, MATLAB.

1. Introduction

Consider a linear regression model $\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \mathbf{e}$, where $\mathbf{y}$ is a vector of n observations, $\mathbf{X}$ is a matrix of covariates, $\boldsymbol{\beta}$ is a vector of p coefficients, and $\mathbf{e}$ is a vector of n random errors. Fitting this model involves two tasks: model selection, where a subset of the coefficients from $\boldsymbol{\beta}$ are included in the model and the rest are excluded (that is, set to zero); and estimation, where the values of the included coefficients are determined. Both of these tasks can be accomplished in one step by adding a penalty term to the regression. For example, Mallows C_p (Mallows 1973), the AIC (Akaike 1973), and the BIC (Schwarz 1978), can all be written as a penalized least-squares problem

$$\tfrac{1}{2n} \left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|_2^2 + \lambda \left\|\boldsymbol{\beta}\right\|_0 \tag{1}$$

where $\|\boldsymbol{\beta}\|_0$ is the number of nonzero coefficients (the L_0 norm) of $\boldsymbol{\beta}$ and λ is a value which may depend on the dispersion in the model and the number of observations. Model selection and estimation are accomplished by finding the $\boldsymbol{\beta}$ which minimizes Equation 1.

Unfortunately, the L_0 norm is non-convex, so it is quite hard to find the minimum of Equation 1. Partly for this reason, it has become popular to use an L_1 penalized regression (the LASSO, Tibshirani 1996) for model selection and estimation:

$$P_\lambda(\boldsymbol{\beta}) = \tfrac{1}{2n} \left\|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\right\|_2^2 + \lambda \left\|\boldsymbol{\beta}\right\|_1 \tag{2}$$

where $\|\boldsymbol{\beta}\|_1 = \sum_i |\beta_i|$ is the L_1 norm of $\boldsymbol{\beta}$. The value of the penalized regression $P_\lambda(\boldsymbol{\beta})$ is not meaningful in the same way that Mallows C_p or the AIC is, so the ideal penalty weight λ is usually determined by minimizing some other criterion, such as cross-validation error. The L_1 penalty can also be added to maximum likelihood estimation; in this case we maximize

$$P_\lambda(\boldsymbol{\beta}) = \tfrac{1}{n} \log \ell(\mathbf{y}; \boldsymbol{\beta}) - \lambda \left\|\boldsymbol{\beta}\right\|_1 \tag{3}$$

(For linear regression, $\log \ell(\mathbf{y}; \boldsymbol{\beta}) = -\frac{1}{2} \|\mathbf{y} - \mathbf{X}\boldsymbol{\beta}\|^2$.)

While the LASSO has plenty of desirable characteristics, it has some potentially undesirable ones too. The LASSO selects coefficients in the model by shrinking *all* coefficients towards zero, so the model with the correct signs and zeros for the coefficients will tend to underfit the data. Any attempt to mitigate the shrinkage by reducing the penalty weight λ will lead the LASSO to add in extra irrelevant coefficients. These problems have led to the proposal of a number of alternative penalties, (for example, SCAD (Fan and Li 2001), MC+ (Zhang 2010; Mazumder, Friedman, and Hastie 2011), FLASH (Radchenko and James 2011), and Relaxo (Meinshausen 2007)) together with their own algorithms and software implementations. Some of these implementations only work with linear regression, or are difficult for the end-user to obtain and use.

It would be useful to be able to carry out penalized model fitting using any penalty, applied to any likelihood, as the problem demands rather than as software availability permits. The **penalized** toolbox is a set of MATLAB (The MathWorks, Inc. 2007) functions which allows you to do this. The toolbox contains functions for penalized maximum likelihood, objects which represent common generalized linear models (least-squares, logistic, multinomial, and poisson), a wide selection of penalty functions, a cross-validation routine, and some plotting functions. Any penalty can be combined with any generalized linear model, and new models and penalties can be added to the toolbox.

This paper describes the toolbox: how to use it, how it works, and how the likelihood models and penalty functions are designed. After a tutorial walkthrough of the toolbox, which shows the sorts of analyses that can be carried out, I outline the maximization algorithm used in the toolbox. Next, I describe the likelihood models and penalty functions, especially how they interface to the core maximization algorithm. Finally, the toolbox is compared with **glmnet** (Friedman, Hastie, and Tibshirani 2010).

2. A tutorial

The **penalized** toolbox is loosely modelled on **glmnet** (Friedman *et al.* 2010) so some of this tutorial may appear familiar to users of that R (R Development Core Team 2008) package. To start, change to the directory containing the toolbox and type

```
> install_penalized
```

This will add the necessary paths to your MATLAB path. **penalized** is pure MATLAB, and uses no MEX files. (You can `uninstall_penalized` later if you want.) This tutorial can be run interactively by typing the command

```
> echodemo jsstutorial
```

To begin, assume we have a set of 0-1 observations **y** together with a covariate matrix X which can be modelled by a logistic regression. To create a logistic model, type

```
> model = glm_logistic(y, X, 'nointercept')
```

(Note that all models in the toolbox are prefixed with `glm_`. The third argument '`nointercept`' specifies that `glm_logistic` should not add an intercept to the design matrix; otherwise an

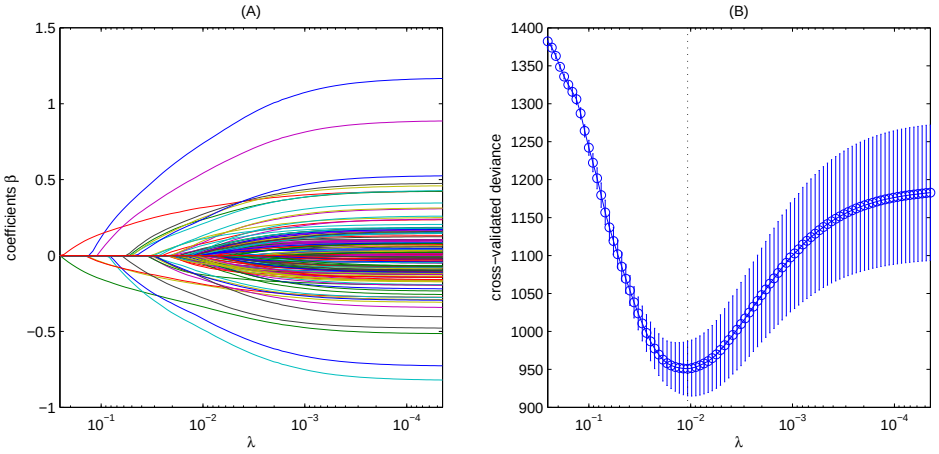

Figure 1: (A) A plot of fitted coefficients against λ on a reversed log scale. (B) A plot of the cross-validation error against λ. The vertical dotted line at $\lambda = 0.0142$ marks the value of λ that yields the smallest cross-validation error.

intercept is added. The intercept is never penalized. To perform a LASSO (L_1 penalized) fit of this model, type

```
> fit = penalized(model, @p_lasso)
```

The first argument to the function `penalized` is the model being fitted. The second argument is a penalty function handle. (The syntax `@function` is like a C function pointer). Here the penalty function is `p_lasso`, one of many penalty functions available in the toolbox. All provided penalty functions are prefixed with `p_`. The function `penalized` then fits an L_1 penalized logistic regression over a range of penalty weights λ.

The flexibility of the toolbox comes from the modularization implied in this function call: *any* model (which conforms to the calling conventions) and *any* penalty function (likewise) can be used.

The returned value, `fit`, is a structure. The field `fit.lambda` contains the sequence of λ values used for penalization. These are automatically selected by the function or manually controlled by the options `lambdamax`, `lambdaminratio`, and `nlambda` which may be passed to `penalized`. (For more information on how `penalized` automatically selects the values of λ, and how this selection process can be influenced or overridden, type `help options`) The field `fit.beta` contains the coefficients: `fit.beta(:,i)` gives the fitted coefficients $\boldsymbol{\beta}$ for the penalty weight `fit.lambda(i)`. If the model has an intercept, it is stored in `fit.beta(1,i)`. The fitted coefficients can be plotted against λ by typing

```
> plot_penalized(fit)
```

yielding the graph shown in Figure 1A. If the model has an intercept, this is omitted from the plot.

One way of selecting the best value of λ is to pick that which minimizes the Akaike Information Criterion (AIC, Akaike (1973)). The AIC can be computed and plotted against lambda by typing

```
> AIC = goodness_of_fit('aic', fit);
> semilogx(fit.lambda,AIC)
```

The function `goodness_of_fit` supplied in the package computes the goodness of fit from the `fit` structure. Instead of '`aic`', you can also pass '`bic`' (the Bayesian Infomation Criterion), '`deviance`', or '`log-likelihood`'. However, the calculation of AIC and BIC assumes that the degrees of freedom is equal to the number of nonzero parameters, which is only know to be true for the LASSO penalty (Zou, Hastie, and Tibshirani 2007).

Alternatively, the best value of λ may be selected by cross-validation. A 5-fold cross-validation of the penalized logistic model can be carried out by typing

```
> cv = cv_penalized(model, @p_lasso, 'folds', 5)
```

The option '`folds`' gives the number of folds; otherwise the arguments to `cv_penalized` are the same as those to `penalized`. (Note that 5 is the default number of folds, so the option wasn't necessary here.) The return structure `cv` contains the results of the cross-validation. The cross-validation error can be plotted against lambda by typing

```
> plot_cv_penalized(cv)
```

The results are shown in Figure 1B. The minimum cross-validation error is obtained at $\lambda = 0.0142$, which is recorded in `cv.minlambda`, and plotted as a vertical dashed line.

Consider now a linear regression model with observations $\mathbf{y}$ and covariate matrix $\mathbf{X}$ and an intercept. We create this by typing

```
> model2 = glm_gaussian(y, X);
```

Since the '`nointercept`' option is omitted, an intercept is added to the model.

Instead of the LASSO penalty, we might try the clipped LASSO (Antoniadis and Fan 2001), as implemented in the toolbox function `p_clipso`. The clipped LASSO penalty function is defined as $clipso(x) = min(|x|, \alpha)$. To use the clipped LASSO with a value of α equal to 0.3, we type

```
> fit = penalized(model2, @p_clipso, 'alpha', 0.3, 'standardize', true)
```

Additional parameters are added as name-value options in the call to `penalized`. Here there are two. The '`alpha`' option specifies the α value for the clipso penalty. The '`standardize`' option effectively scales the columns of `X` to have equal norms. This is useful when the penalty function is not scale invariant (which is unfortunately the case for many penalty functions). Standardization is reversed when the fitted coefficients are calculated. The results of this fit can again be plotted using `plot_penalized(fit)`, yielding the graph in Figure 2A. The intercept is omitted from the plot.

Instead of a single value for α, a range of values can be efficiently fitted in a single function call by setting '`alpha`' to an array:

```
> fit = penalized(model2 ,@p_clipso, 'alpha', [inf, 1, 0.5, 0.3, 0.05])
```

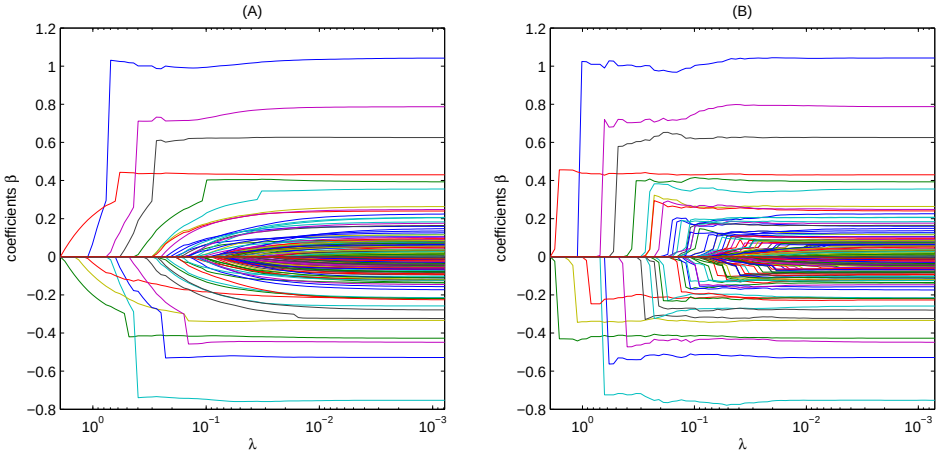

Figure 2: (A) A plot of fitted coefficients using the clipso penalty for parameter $\alpha = 0.3$ against $\log \lambda$. You can see the values of the coefficients "jump" when they reach the clipping parameter α. (B) A plot of fitted coefficients for clipso parameter $\alpha = 0.05$ against $\log \lambda$.

(An infinite value for α makes `p_clipso` behave the same as `p_lasso`.) The returned `fit` structure holds the coefficients for all values of penalty weight λ, and all the values of α specified in the function call. That is, `fit.beta(:, i, j)` holds the fitted coefficients $\boldsymbol{\beta}$ for the penalty weight `fit.lambda(i)` and the penalty parameter `fit.alpha(j)`, where in this case `fit.alpha` will be equal to `[inf, 1, 0.5, 0.3, 0.05]`.

We can plot all these coefficients using `plot_penalized(fit)`. This interactively displays the fitted coefficients against λ for each value of α, pausing after each plot. A specific value or values of α can be picked out for plotting by typing

```
> plot_penalized(fit, 'slice', 5)
```

This plots the fitted coefficients $\boldsymbol{\beta}$ against λ for the 5th α value (the slice), which is equal to 0.05. The result is shown in Figure 2B. The intercept is omitted from this plot. We can pick the best value of λ *and* α simultaneously by cross-validation. Typing

```
> cv = cv_penalized(model2, @p_clipso, 'alpha', [inf, 1, 0.5, 0.3, 0.05], ...
       'folds', 3)
```

will do a three-fold cross-validation of the model over a range of λ and the specified values of α. The results can be plotted with `plot_cv_penalized(cv)`, as shown in Figure 3A. In this case, the plot superimposes the cross-validation error versus λ curve for each value of α on the same axes. Error bars are omitted from this plot. The minimum cross-validation error is attained for $\alpha = 0.05$ and $\lambda = 0.1895$ (which are stored in `cv.minalpha` and `cv.minlambda`). Cross-validation errors for specific values of α (with error bars) can also be plotted. For example

```
> plot_cv_penalized(cv, 'slice', [1, cv.minalpha], 'errorbars', 'on')
```

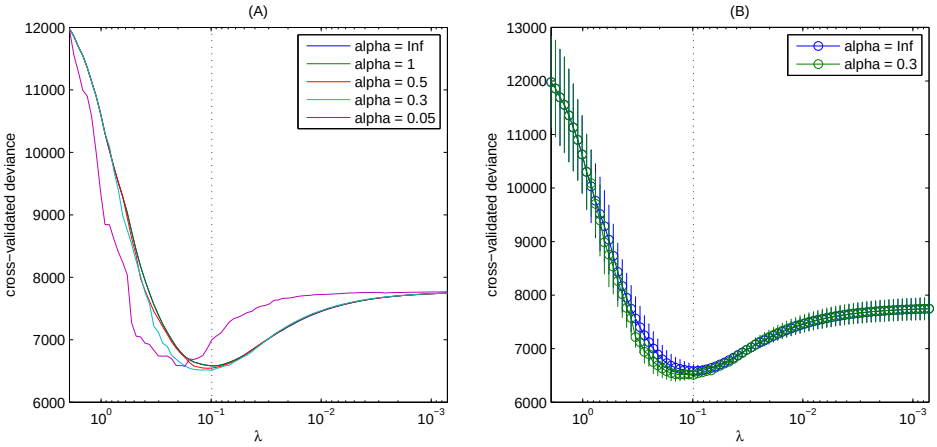

Figure 3: (A) A plot of the cross-validation error versus λ for all values of the penalty parameter α. Error bars are omitted for clarity in this plot. (B) A plot of cross-validation error versus λ for two specific values of the `clipso` penalty parameter α. The blue plot shows cv error for $\alpha = \infty$, and the green plot for $\alpha = 0.05$, which has the best cross-validation error.

will plot the cross-validation error for the 1st value of α (∞) and the cross-validation error for the best value of α, saved in `cv.minalpha`. This is shown in Figure 3B. In this case the best cross-validation error for $\alpha = 0.05$ isn't distinguishable from the best one produced by the lasso ($\alpha = \infty$).

If you have a custom penalty function `mypenalty`, that requires a parameter γ, it can be used by typing

```
> fit = penalized(model2, @mypenalty, 'gamma', 2)
```

and the results can be plotted, as before. The function `mypenalty` must follow the calling conventions outlined in Section 5 below. Cross validation works just as before:

```
> cv  =  cv_penalized(model2, @mypenalty, 'gamma', 1:5, 'folds', 3)
```

Here the cross-validation is done for a range of γ values from 1 to 5. In this case, the field `cv.mingamma` says which value of γ yielded the smallest cross-validation error.

Similarly, if you have developed a custom statistical model `mymodel`, it can be fitted with any penalty by typing

```
> fit = penalized(mymodel, @mypenalty, 'gamma', 2)
```

Cross-validation and plotting work as before. The custom model `mymodel` must be a MATLAB object with the methods outlined in Section 4.1 below.

3. The maximization algorithm

The algorithm used in the **penalized** toolbox is Fisher scoring over an active set with orthant projection (Schmidt, Fung, and Rosales 2009; Park and Hastie 2007). It is described here

simply to indicate how the likelihood model and the penalty function interact with it, as there is nothing original in this implementation.

We consider only penalties $\pi(\boldsymbol{\beta})$ that are a weighted sum of coordinate penalty functions, so that $\pi(\boldsymbol{\beta}) = \sum_{i=1}^{p} w_i \pi_i(\beta_i)$, where π_i is the penalty function for the i-th coefficient β_i, and w_i are the penalty weights. We wish to maximize the penalized likelihood

$$P_\lambda(\boldsymbol{\beta}) = \tfrac{1}{n} \log \ell(\mathbf{y}; \boldsymbol{\beta}) - \lambda \pi(\boldsymbol{\beta}) \tag{4}$$

Define the score $s(\boldsymbol{\beta})$ as the gradient[1] of the likelihood, $s(\boldsymbol{\beta}) = d \log \ell(\mathbf{y}; \boldsymbol{\beta})/d\boldsymbol{\beta}$. Define $\pi'(\boldsymbol{\beta}) = d\pi(\boldsymbol{\beta})/d\boldsymbol{\beta}$ as the gradient of the penalty function (with elements $w_i d\pi_i(\beta_i)/d\beta_i$). Define $\mathbf{H}(\boldsymbol{\beta})$ as the Hessian of the likelihood $\ell(\mathbf{y}; \boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$, and Π as the Hessian of the penalty $\pi(\boldsymbol{\beta})$. This is simply a diagonal matrix with entries $w_i d^2\pi_i(\beta_i)/d\beta_i^2$.

The Newton iteration

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t - (\tfrac{1}{n}\mathbf{H}(\boldsymbol{\beta}^t) - \lambda\Pi)^{-1}(\tfrac{1}{n}s(\boldsymbol{\beta}^t) - \lambda\pi'(\boldsymbol{\beta}^t)) \tag{5}$$

is used to find the next estimate $\boldsymbol{\beta}^{t+1}$ from the existing estimate $\boldsymbol{\beta}^t$. In Fisher scoring, the Hessian $\mathbf{H}$ is replaced by the negative of the Fisher Information matrix, $-\mathbf{F}$, to give

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t + (\tfrac{1}{n}\mathbf{F} + \lambda\Pi)^{-1}(\tfrac{1}{n}s(\boldsymbol{\beta}^t) - \lambda\pi'(\mathbf{b}^t)) \tag{6}$$

This iteration is rapidly convergent when $\boldsymbol{\beta}^t$ is close to the maximum, but might be poor far from it. Problematic convergence can be improved using a Levenburg-Marquardt adjustment, adding in the term $\omega\text{diag}(\frac{1}{n}\mathbf{F})$, to yield

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{\beta}^t + (\tfrac{1}{n}\mathbf{F} + \lambda\Pi + \omega\text{diag}(\tfrac{1}{n}\mathbf{F}))^{-1}(\tfrac{1}{n}s(\boldsymbol{\beta}^t) - \lambda\pi'(\boldsymbol{\beta}^t)) \tag{7}$$

The Levenburg-Marquardt weight ω is iteratively adjusted using a trust-region algorithm. If there is some improvement in the penalized log-likelihood $P_\lambda(\boldsymbol{\beta}^{t+1}) - P_\lambda(\boldsymbol{\beta}^t)$ then ω is decreased or kept the same, depending on the size of the improvement. However, if there is no improvement in the penalized log likelihood, then ω is repeatedly increased until some improvement occurs. If, after many attempts with increasing ω, there is still no improvement in the penalized log-likelihood, then the iteration halts.

However, (7) assumes that the penalty functions are differentiable. Most interesting penalties are not, so the algorithm must be changed to deal with this. An active set algorithm is used, as follows.

The vector $\boldsymbol{\beta}$ is divided into two parts, the 'active' vector $\boldsymbol{\beta}_A$, where differentiability of the penalty holds, and the 'inactive' vector $\boldsymbol{\beta}_{\sim A}$, where it doesn't. The algorithm assumes that the only singularity of the coordinate penalty functions π_i is at zero, so the active vector is simply all the non-zero entries of $\boldsymbol{\beta}$, and the inactive vector is all the zero entries.

Nondifferentiability at zero is handled using subderivatives. At a nondifferentiable point β of the penalty π, let $\pi^-(\beta)$ be the left derivative and $\pi^+(\beta)$ be the right derivative. The *subdifferential* is the interval $[\pi^-(\beta), \pi^+(\beta)]$, and a *subderivative* is any value in the subdifferential.

Each modified iteration step proceeds in two parts:

[1] The notation $df/d\mathbf{x}$, where f is a scalar, is the gradient $(\partial f/\partial x_1, \ldots, \partial f/\partial x_n)$ with respect to $\mathbf{x}$. The notation $d\mathbf{f}/d\mathbf{x}$, where $\mathbf{f}$ is a vector, is the Jacobian matrix with elements $J_{i,j} = \partial f_i/\partial x_j$.

(A) Gradient ascent on the inactive vector: Any elements β_i in $\boldsymbol{\beta}_{\sim A}$ for which

$$\tfrac{1}{n}s(\beta_i) \notin [\lambda w_i \pi_i^-(\beta_i),\ \lambda w_i \pi_i^+(\beta_i)]$$

(where $s(\beta_i)$ is the i-th element of the score) do not satisfy the first-order optimality conditions for the maximum, and so are candidates for addition to the active vector. Theoretically, all elements β_i in $\boldsymbol{\beta}_{\sim A}$ for which the above holds could be added to the active vector, but in practice the algorithm is more stable if, at every iteration, only the few elements which violate the optimality conditions the most are added (Perkins, Lacker, and Theiler 2003). For those few elements, we set $\beta_i = \beta_i + \epsilon s(\beta_i)$, for some small ϵ. These elements then enter the active vector.

(B) Fisher update on the active vector: Holding the inactive vector fixed, use the update

$$\boldsymbol{\beta}_A^{t+1} = \boldsymbol{\beta}_A^t + (\tfrac{1}{n}\mathbf{F}_A + \lambda\Pi_A + \omega\mathrm{diag}(\tfrac{1}{n}\mathbf{F}_A))^{-1}(\tfrac{1}{n}s(\boldsymbol{\beta}_A^t) - \lambda\pi'(\boldsymbol{\beta}_A^t)) \tag{8}$$

where $\mathbf{F}_A$ and Π_A are the information and penalty matrices restricted to elements of the active vector. This step may only be valid when $\boldsymbol{\beta}_A^t$ and $\boldsymbol{\beta}_A^{t+1}$ are in the same orthant. Thus, $\boldsymbol{\beta}_A^{t+1}$ is projected onto the closest point in the orthant containing $\boldsymbol{\beta}_A^t$ for which twice-differentiability does hold (Andrew and Gao 2007), giving us the update rule

$$\boldsymbol{\beta}_A^{t+1} = Proj_A\{\boldsymbol{\beta}_A^t + (\tfrac{1}{n}\mathbf{F}_A + \lambda\Pi_A + \omega\mathrm{diag}(\tfrac{1}{n}\mathbf{F}_A))^{-1}(\tfrac{1}{n}s(\boldsymbol{\beta}_A^t) - \lambda\pi'(\boldsymbol{\beta}_A^t))\} \tag{9}$$

where $Proj_A$ is the orthant projection operator, which simply zeros all elements of $\boldsymbol{\beta}_A^{t+1}$ which have a different sign from $\boldsymbol{\beta}_A^t$. Elements of the active set which are zeroed then join the inactive set. If the projected step doesn't lead to a reduction in the penalized likelihood $P_\lambda(\boldsymbol{\beta})$, then the trust-region procedure shrinks the step until it does.

3.1. Warm starts

Because we are interested in the value of the coefficients $\boldsymbol{\beta}$ over a range of penalty weights λ, a continuation process is used to estimate them efficiently: the best fitted value $\boldsymbol{\beta}_k^*$ for a given penalty weight λ_k is used as the initial value of the iterations with the next penalty weight λ_{k+1}.

A continuation process is also used when there are a range of penalty parameters to fit (e.g., α in `clipso`). Writing $\boldsymbol{\beta}_{k,j}^*$ to be the best fitted coefficients for penalty weight λ_k and penalty parameter j, the initial value $\boldsymbol{\beta}_{k,j}^0$ for the k-th value of λ and the j-th value of the penalty parameter can be either:

1. The best fit from the previous value of the penalty parameter, $\boldsymbol{\beta}_{k,j-1}^*$. This option is selected by typing `penalized( ..., 'warmstart', 'relax')`. You can think of this as fixing a particular weight λ, then "relaxing" the penalty parameter from the first value specified to the last. This is the default option.

2. The best fit from the previous value of penalty weight λ, $\boldsymbol{\beta}_{k-1,j}^*$. This option is selected by typing `penalized( ..., 'warmstart', 'lambda')`. You can think of this as fixing a particular penalty parameter at j, then fitting successive values of the weight λ.

3. Use both warm starts above and pick the best. This option is selected by typing `penalized( ..., 'warmstart', 'both')`. Obviously this takes twice as long.

4. The likelihood

The maximization algorithm needs the score vector $s(\boldsymbol{\beta})$ and Fisher information matrix $\mathbf{F}$ from the likelihood model. For some likelihoods – generalized linear models (Nelder and Wedderburn 1972; McCullagh and Nelder 1989) – these are relatively simple to compute. In this section, we briefly review the construction of generalized linear models and specify how they interface with the maximization algorithm.

A generalized linear model has a number of independent observations y_i with expected values $\mathsf{E}(y_i) = \mu_i$. Each observation has an associated covariate vector $\mathbf{x}_i = \{x_{i1}, x_{i2}, ...x_{ip}\}$, which are the rows of a covariate matrix $\mathbf{X}$. The linear predictor is a weighted sum of covariates, $\eta_i = \mathbf{x_i}\boldsymbol{\beta}$ where $\boldsymbol{\beta}$ is the vector of coefficients that we wish to estimate. The linear predictor is related to the expected value of y_i by the link function $f(\mu_i) = \eta_i$.

Because the observations are independent, the log-likelihood of the data $\ell(\mathbf{y};\boldsymbol{\beta})$ with respect to the coefficient vector $\boldsymbol{\beta}$ is the sum of the log-likelihoods of the observations: $\ell(\mathbf{y};\boldsymbol{\beta}) = \sum_{i=1}^{n} \ell_i(y_i;\boldsymbol{\beta})$. The derivative of the log-likelihood with respect to the j-th element of $\boldsymbol{\beta}$ is:

$$s(\boldsymbol{\beta})_j = \frac{\partial \ell(\mathbf{y};\boldsymbol{\beta})}{\partial \boldsymbol{\beta}_j} = \sum_{i=1}^{n} \frac{d\ell_i(y_i)}{d\mu_i}\frac{d\mu_i}{d\eta_i}\frac{\partial \eta_i}{\partial \boldsymbol{\beta}_j} = \sum_{i=1}^{n} \frac{d\ell_i(y_i)}{d\mu_i}\frac{d\mu_i}{d\eta_i}x_{i,j} \tag{10}$$

In vector notation, $s(\boldsymbol{\beta}) = \mathbf{X}^\top\mathbf{D}\mathbf{m}$ where $\mathbf{m}$ is a vector of derivatives with elements $m_i = d\ell_i(y_i)/d\mu_i$ and $\mathbf{D}$ is a diagonal matrix with elements $D_{i,i} = d\mu_i/d\eta_i$.

The Fisher information matrix $\mathbf{F}$ is given by

$$\mathbf{F} = \mathsf{E}((\nabla\ell(\mathbf{y};\boldsymbol{\beta}))(\nabla\ell(\mathbf{y};\boldsymbol{\beta}))^\top) = \mathsf{E}((\mathbf{X}^\top\mathbf{D}\mathbf{m})(\mathbf{X}^\top\mathbf{D}\mathbf{m})^\top) = \mathbf{X}^\top\mathbf{D}\mathsf{E}(\mathbf{m}\mathbf{m}^\top)\mathbf{D}\mathbf{X} \tag{11}$$

Because the observations are independent, $\mathsf{E}(\mathbf{m}\mathbf{m}^\top)$ is a diagonal matrix with elements $\mathsf{E}(m_i^2) = \mathsf{E}((d\ell_i(y_i)/d\mu_i)^2)$. Calling this matrix $\mathbf{V}$, the information matrix can be written as $\mathbf{F} = \mathbf{X}^\top\mathbf{D}\mathbf{V}\mathbf{D}\mathbf{X}$. The information matrix over the active set is just $\mathbf{F}_A = \mathbf{X}_A^\top\mathbf{D}\mathbf{V}\mathbf{D}\mathbf{X}_A$, where $\mathbf{X}_A$ is the covariate matrix restricted to those columns which are in the active set.

4.1. Interface to the maximization algorithm

The score and information matrix could be supplied by a function, which is typically what occurs in maximization routines (see for example, the MATLAB function `fminunc` in the optimization toolbox). However, because the likelihood depends on a substantial amount of data (the observations $\mathbf{y}$ and covariates $\mathbf{X}$), and needs additional book-keeping functions, it is best implemented as an object, which has methods that can be called by the maximization algorithm as needed. The most straightforward interface would be for the likelihood object to supply the log-likelihood $\log \ell(\mathbf{y};\boldsymbol{\beta})$, the score $s(\boldsymbol{\beta})$ and the information matrix $\mathbf{F}(\boldsymbol{\beta})$ as requested. However, the full information matrix could be extremely large, even when the active set is small. A more efficient interface is for the likelihood object to supply the components needed to compute the information matrix, and allow the maximization algorithm to assemble them in an efficient way.

Thus, if `model` is a likelihood object representing a generalized linear model, the log-likelihood at $\boldsymbol{\beta}$ is computed by a call to the method `logl`:

```
> L = logl(model, beta)
```

The components $\mathbf{m}$, $\mathbf{D}$, $\mathbf{V}$, and $\mathbf{X}$ as a function of $\boldsymbol{\beta}$ are returned by a call to the method `scoring`:

```
> [L, m, D, V, X] = scoring(model, beta)
```

Here `L` is the log-likelihood again; it's efficient to return it at the same time as the other quantities $\{\mathbf{m}, \mathbf{D}, \mathbf{V}, \mathbf{X}\}$. The diagonal matrices $\mathbf{D}$ and $\mathbf{V}$ are returned as column vectors; when all diagonal entries in $\mathbf{D}$ or $\mathbf{V}$ are the same, these can be returned as scalars.

Though the meaning of the elements in the tuple $\{\mathbf{m}, \mathbf{D}, \mathbf{V}, \mathbf{X}\}$ are specified for generalized linear models, the maximization algorithm doesn't care; it requires only that the information matrix $\mathbf{F}_A$ is equal to $\mathbf{X}_A^\top \mathbf{D}\mathbf{V}\mathbf{D}\mathbf{X}_A$ and the score $s(\boldsymbol{\beta})$ is equal to $\mathbf{X}^\top \mathbf{D}\mathbf{m}$. For example, the likelihood object could return the tuple $\{\mathbf{D}\mathbf{m}, 1, \mathbf{D}\mathbf{V}\mathbf{D}, \mathbf{X}\}$ or $\{\mathbf{m}, 1, \mathbf{V}, \mathbf{D}\mathbf{X}\}$ instead of $\{\mathbf{m}, \mathbf{D}, \mathbf{V}, \mathbf{X}\}$, as they yield exactly the same score and information matrix.

As well as the above two methods `logl` and `scoring`, the likelihood object must also have the following methods:

`obj = constructor(...)`	The constructor creates the likelihood object. All of the provided constructors will add an intercept unless a '`nointercept`' option has been given. However, this is just a convention.
`p = property(model)`	returns a structure containing a number of model properties. The structure should contain fields `n` and `p` for the number of rows and columns in $\mathbf{X}$ respectively; `nobs` for the number of observations, which may be different from `n` (e.g., in `glm_logistic`); `intercept` which is the empty matrix if no intercept, and 1 if an intercept has been inserted as column 1; and `colscale` which gives the L_2 norms of the columns of $\mathbf{X}$ (used for standardization).
`p = property(model,'name')`	returns a specific property of the model indicated by the name.
`beta = initial(model)`	returns a suitable initial value for $\boldsymbol{\beta}$, usually just zeros.
`s = sample(model, index)`	returns a subset `s` of the model having observations in the `index`. This is used in cross-validation.
`beta = project(model, beta)`	projects $\boldsymbol{\beta}$ onto the allowable domain of the model, when the domain of $\boldsymbol{\beta}$ is restricted. Otherwise, it returns $\boldsymbol{\beta}$ unchanged.

Further details of these methods can be found by typing `help models`. The toolbox provides `glm_gaussian`, `glm_logistic`, `glm_poisson`, and `glm_multinomial` class constructors. These all inherit from a base class `glm_base`, which provides many of the above methods.

5. Penalties

The third component of the **penalized** toolbox is the penalty function $\pi(\boldsymbol{\beta}) = \sum_{i=1}^{p} w_i \pi_i(\beta_i)$. If the weights w_i in the penalty are different from 1, they can be specified in a call to `penalized` by including the `penaltywt` option. For example, `penalized(model, @penalty, 'penaltywt', w)` uses elements of the vector `w` as the penalty weights.

A penalty function is useful in a model selection role by inducing sparseness in the coefficient vector $\boldsymbol{\beta}$. The ability of a penalty function to induce sparseness arises from the discontinuous derivative of the penalty at zero. Coefficients β_i satisfy the first-order optimality conditions when

$$\tfrac{1}{n} s(\beta_i) \in [\lambda w_i \pi_i^-(\beta_i),\ \lambda w_i \pi_i^+(\beta_i)]$$

What this implies is that coefficients β_i are "trapped" at the singularity (Fan and Li 2001) until their score $\frac{1}{n}s(\beta_i)$ exceeds the maximum subderivative multiplied by λw_i. Thus penalties which have a singularity at zero are sparsity inducing, because they trap coefficients at zero.

Because the singularity at zero is the only useful for inducing sparsity, the maximization algorithm assumes that it is the *only* singularity. However, some penalties also have discontinuous derivatives away from zero (such as `p_clipso`). These discontinuities don't induce sparsity, and are ignored by the maximization algorithm. This does not cause a problem with `p_clipso`, but other penalties with non-differentiable points away from zero might possibly fail. The subderivative at zero also needs to be finite; otherwise coefficients will be permanently trapped there.

5.1. Supplied penalty functions

The penalized toolbox supplies the following penalty functions:

Adaptive (Zou 2006): The adaptive LASSO is $\pi_i(\beta_i) = |\beta_i|/|\hat{\beta}_i|^\gamma$, where $\hat{\beta}_i$ is a consistent estimate of β_i, such as the ordinary least-squares estimate. The adaptive LASSO is discussed further in Section 6.2.

Concave PF (Nikolova 2000, p. 653): The concave PF penalty is given by $\pi_i(\beta_i) = k|\beta_i|/(k + |\beta_i|)$. This penalty behaves like the LASSO when $k = \infty$, and like the L_0 penalty when $k = 0$.

Clipso (Antoniadis and Fan 2001): The clipped LASSO is $\pi_i(\beta_i) = \min(|\beta_i|, \alpha)$. The scaled clipso is $\pi_i(\beta_i) = \min(|\beta_i|, \lambda\alpha)$.

Elastic (Zou and Hastie 2005): This is a linear combination of LASSO and Ridge, $\pi_i(\beta_i) = \alpha|\beta_i| + (1-\alpha)\beta_i^2$.

FLASH (Radchenko and James 2011): The FLASH algorithm is not defined as a penalized optimization, but it has an implied penalty. The FLASH penalty, with parameter δ, is equal to $|\beta_i|$ when $\beta_i = 0$, and is equal to $(1-\delta)|\beta_i|$ when $\beta_i \neq 0$. The equivalence between the FLASH algorithm and this penalty is shown in Appendix A. To use FLASH as described in Radchenko and James (2011), you must call `penalized(model, @p_flash, ..., 'warmstart', 'lambda')` to get the correct warm-starts.

LASSO (Tibshirani 1996): The LASSO, or L_1 penalty, is $\pi_i(\beta_i) = |\beta_i|$.

Lq : The Lq penalty is just the Lq norm, $\pi_i(\beta_i) = |\beta_i|^q$. When q is less than 1, this penalty will trap all elements of $\boldsymbol{\beta}$ in the inactive set because the subderivatives are infinite. However, it can be used in `penalized` when a range of different q is provided. For example the call `penalized(model, @p_Lq, 'q', [1 0.8 0.6 0.4 0.2 0])` will work, because the first value of q is just the lasso, and subsequent values of q use the lasso solution as a warm start.

MC+ (Zhang 2010; Mazumder *et al.* 2011): The MC+ penalty is easiest defined by its derivative $\pi_i'(\beta_i) = sign(\beta_i)(1 - |\beta_i|/(\alpha))_+$ when $\beta_i \neq 0$, and $\pi_i^{sub}(\beta_i) = [-1, 1]$ when $\beta_i = 0$.

None : This penalty function doesn't penalize, for cases where unpenalized maximum likelihood is needed. So $\pi_i(\beta_i) = 0$. When using this penalty, the convergence criteria for `penalized` may need to be tightened. See `help options`.

Ridge (Hoerl and Kennard 1970): The ridge penalty is $\pi_i(\beta_i) = \beta_i^2$. When using the ridge penalty, you must supply a starting value for λ, using the `lambdamax` option, e.g., `penalized(model, @p_ridge, 'lambdamax', 1)`.

SCAD (Fan and Li 2001): The Smoothly Clipped Absolute Deviation penalty is easiest defined by its derivative

$$\pi_i'(\beta_i) = \begin{cases} sign(\beta_i) & |\beta_i| < \lambda \\ sign(\beta_i)(a\lambda - |\beta_i|)/((a-1)\lambda) & \lambda \leq |\beta_i| < a\lambda \\ 0 & |\beta_i| \geq a\lambda \end{cases}$$

SCAD behaves like the lasso when $|\beta_i| < \lambda$, does not penalize when $|\beta_i| \geq a\lambda$, and smoothly transitions between these two behaviours when $\lambda \leq |\beta_i| < a\lambda$. The parameter a must be greater than 2.

Other penalty functions can be defined and used so long as they adhere to the calling conventions given next.

5.2. Interface to the maximization algorithm

In the maximization algorithm, the penalty function must supply, at different times, the individual penalties $\pi_i(\beta_i)$, the derivatives $\pi_i'(\beta_i)$, the subdifferential $[\pi_i^-(\beta_i),\ \pi_i^+(\beta_i)]$, and the second derivatives $\pi_i''(\beta_i)$, when requested. Switching between these is accomplished with a mode parameter. Any additional parameters needed by the penalty function are passed in as fields in an options structure. For example, the call `penalized(model, @clipso, 'alpha', 0.3)` will create an options structure which contains a field `options.alpha` equal to 0.3. This options structure is then passed to the penalty function.

The penalty function takes four arguments - a mode string, a coefficient vector, the current value of λ, and an options structure `options`, with any extra parameters as fields. The following call patterns are expected:

```
> p = penalty('', beta, lambda, options)
```

An empty mode string asks the penalty function to return the penalty values for a coefficient vector `beta`; `p(i)` is the penalty $\pi_i(\beta_i)$ for coefficient `beta(i)`. The total penalty is `lambda*sum(w.*p)` where `w` is a vector of weights, usually 1. The vector of weights can be specified by the option `penalized(... 'penaltywt', w)`.

```
. d = penalty('deriv', beta, lambda, options)
```

This returns a vector `d` of derivatives, where `d(i)`=$d\pi_i(\boldsymbol{\beta}_i)/d\boldsymbol{\beta}_i$. Elements of `d` in the inactive set are ignored, so any value can be returned for them. For elements of `d` in the active set, and where the derivative is discontinuous, return either endpoint of the subdifferential or the average of the endpoints.

```
> [lo,hi] = penalty('subdiff', beta, lambda, options)
```

This returns the subdifferential intervals for elements `beta(i)`. The return values should be `lo(i)`= $\pi_i^-(\beta_i)$ and `hi(i)`=$\pi_i^+(\beta_i)$. Elements of `lo` and `hi` that are in the active set are ignored, so any value can be used there. If all subdifferential intervals in the inactive set are the same (which is usually the case), then `lo` and `hi` can be scalars rather than vectors.

```
> p2 = penalty('2ndderiv', beta, lambda, options)
```

This returns the vector of second derivatives of the penalty for the parameter `beta`. If all second derivatives are the same, `p2` can be a scalar. Elements of `p2` that are not in the active set are ignored.

```
> tf = penalty('project', beta, lambda, options)
```

This returns true if the orthant projection $Proj_A$ is required for the coefficient vector β. This is true for most penalties; exceptions are `ridge` and `none`.

For example, suppose we want to create a new penalty $\pi(\beta) = log(1 + \alpha|\beta|)$, which we will call `abslog`. The penalty will be called with a specific parameter $\alpha = 1$ by typing `penalized(model,@abslog,'alpha',1)`. The function `penalized` will put α into an options structure which will be passed to our penalty function as the last parameter.

Thus the first two lines of our new penalty can be

```
> function [x,y]=abslog(mode, beta, lambda, options)
> alpha = options.alpha;
```

When the mode parameter is '', we return the penalty values in the `x` variable

```
> x = log(1+alpha*abs(beta));
```

When the mode is '`deriv`', we return the derivative.

```
> x = alpha*sign(beta)./(1+alpha*abs(beta));
```

When the mode is '`subdiff`' we return the endpoints of the subdifferential in `x` and `y`:

```
> x = -alpha; y = alpha;
```

When the mode is '2ndderiv' we return the second derivative

```
> x = -alpha^2./(1+alpha*abs(beta));
```

Finally, when the mode is 'project' we return x=true;, because this penalty requires orthant projection.

6. Related algorithms

The **penalized** toolbox can also be used to implement some other penalized likelihood algorithms.

6.1. The relaxed LASSO

The relaxed lasso (Meinshausen 2007) is a way of successively reducing the shrinkage over the active set of parameters. The relaxed lasso is defined as

$$P_\lambda(\boldsymbol{\beta}) = \log \ell(\mathbf{y}; \boldsymbol{\beta}) - \phi\lambda \left\|\boldsymbol{\beta}\right\|_1 \tag{12}$$

where ϕ is the relaxation parameter. Initially, $\phi = 1$, but after the coefficients for a given λ have been determined, ϕ is relaxed towards zero over the non-zero coefficients, while holding the other coefficients at zero. This is simply the FLASH penalty with $\delta = 1 - \phi$, so we can use the FLASH penalty to implement relaxed lasso (relaxo) with the following call

```
> fit = penalized(model, @p_flash, 'delta', 0:0.1:1, 'warmstart', 'relax')
```

(Note that 'warmstart', 'relax' is the default and can be omitted.) In this case, the FLASH parameter $\delta = 1 - \phi$ is relaxed in increments of 0.1. The penalized function fits a full sequence of λ for $\delta = 0$ – i.e., a LASSO fit – then for each fit, relaxes δ from 0 through to 1.

6.2. The adaptive LASSO

The penalized function allows you to set individual penalty weights for each coefficient by adding a 'penaltywt' option. The weighted penalized likelihood

$$P_\lambda(\boldsymbol{\beta}) = \log \ell(\mathbf{y}; \boldsymbol{\beta}) - \lambda \sum_{i=1}^{p} w_i |\beta_i| \tag{13}$$

can be run with the call

```
> fit = penalized(model, @p_lasso, 'penaltywt', w)
```

where w is a vector of the penalty weights w_i (the penalty weight for any intercept is always forced to be zero).

The adaptive lasso uses a particular set of penalty weights which ensure an oracle property and near minimax estimation (Zou 2006). The adaptive lasso could be implemented in penalized by setting $w_i = 1/|\hat{\beta}_i|^\gamma$ where $\hat{\beta}_i$ is a consistent estimate of β_i, such as the ordinary least

squares estimate. Unfortunately, this approach does not permit us to use cross-validation to find the best value of the power γ, since γ needs to be a parameter for this to work.

Thus, we pass the adaptive lasso weights as a separate option in the call to the `penalized` routine. The separate option is called '`adaptivewt`'. The adaptive LASSO can be called in two ways. If only one value of gamma is used, say 0.5, then the adaptive LASSO is invoked as

```
> fit = penalized(model, @p_adaptive, 'gamma', 0.5, 'adaptivewt', {beta_ols})
```

where `beta_ols` is the vector of adaptive LASSO weights. The adaptive weight vector is enclosed in a cell because when `penalized` reads the parameters, it assumes that any non-standard option (that is, one which is not described in `help options`)whose value is an array must be a penalty parameter which should then be 'relaxed' over. As the adaptive weights are not a relaxation parameter, they are enclosed in a cell to avoid this misinterpretation. If multiple values of gamma are used, then the call is, for example,

```
> fit = penalized(model,@p_adaptive, 'gamma', 1:-0.2:0.01, 'adaptivewt', {beta_ols})
```

Again the adaptive weights must be enclosed as a cell so that `penalized` interprets it correctly.

7. Performance

Flexibility and extensibility were the overriding design concerns for the **penalized** toolbox. Performance was not completely ignored, however, and while **penalized** is slower than **glmnet**, it completes in reasonable times. Table 1 gives some representative running times for **penalized** compared to **glmnet**. The timings for **penalized** were obtained using MATLAB 2007b (32 bit) running on a Samsung EP300E5C laptop (Core i3 2.4GHz, 6GB memory, using mains power) under Windows 7(64bit). The timings for **glmnet** were obtained on the same machine using R (64 bit). Each timing is an average of 15 runs. Each run fitted a sequence of 100 lambdas on the same data set.

The design matrices $\mathbf{X}$ used in the timings were randomly generated with uncorrelated columns; however, introducing correlations did not change the comparisons very much and so are not shown. The true β coefficients oscillated between positive and negative values, with an exponential decay on their magnitude. The rate of decay was such that the 7th coefficient was half the size of the first. The set of λ values was selected by **glmnet** and then used by **penalized**, to ensure that they both computed the same number of parameters.

penalized is rarely more than a few seconds slower than **glmnet**, and most of the time the difference isn't really noticeable. If using a parameterized penalty (e.g., `p_clipso`, `p_Lq`), the time taken increases with the number of parameters. For example

```
> penalized(model2, @p_clipso, 'alpha', [inf 1 0.5 0.3 0.1])
```

will take about 5 times longer than

```
> penalized(model2, @p_clipso, 'alpha', 0.3)
```

n	1000	5000	10000	100000	100	100	100
p	100	100	100	100	1000	5000	10000
Gaussian model							
glmnet	0.02	0.06	0.10	0.81	0.04	0.11	0.20
penalized	0.28	0.62	1.08	8.09	0.43	0.93	1.58
Logistic model							
glmnet	0.15	0.62	1.19	11.20	0.06	0.14	0.25
penalized	0.53	1.49	2.62	23.02	0.78	1.47	1.97

Table 1: Comparison of average timings, in seconds, for **penalized** in MATLAB and **glmnet** in R. All times are in seconds and are an average of 15 runs. Each column is a different size of problem given by the number of observations n and the number of parameters p.

Likewise, cross-validation time increases linearly with the number of folds.

7.1. Accuracy

The speed of **penalized** depends on the options controlling convergence at each level of λ. The default convergence criteria were chosen to maximize speed while keeping the coefficient estimates close to those provided by **glmnet**. Generally, the difference between the estimates provided by **glmnet** and **penalized** were less than 0.5% of the norm of the coefficient vector. Figure 4(A) compares the estimated coefficients from **penalized** (solid lines) and **glmnet** (dots) for one run ($n = 1000, p = 100$) using a logistic model. Other models are similar. The estimates from the two packages follow each other closely. The differences in the coefficients are plotted against λ in Figure 4(B). The jags in this plot are due to different convergence criteria. In **glmnet** the algorithm converges when all coefficients have a small enough change, while in **penalized** it converges when the vector of coefficients has a small enough change. This means that variables which enter the active set in **penalized** may not move very much until the next lowest λ is used.

8. Conclusion

penalized is a flexible and efficient MATLAB toolbox for using and exploring penalized regression with generalized linear models. It allows the user to use any penalty with any generalized linear model. The toolbox can be extended to include other log-likelihood models and other penalties than those provided, making it simpler to explore the performance of any model or penalty that can be coded to the toolbox API. The toolbox also has the option to select the underlying maximization algorithm, so future versions may include a faster maximization algorithm for the gaussian model.

References

Akaike H (1973). "Information Theory and an Extension of the Maximum Likelihood Prin-

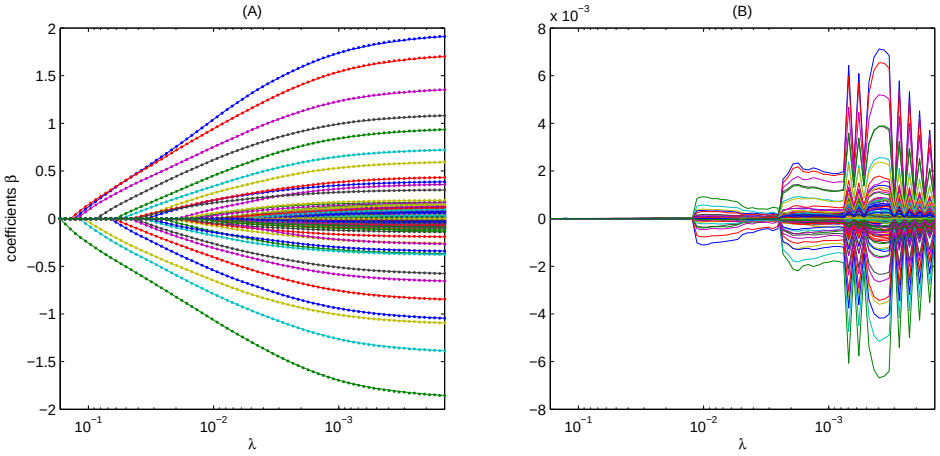

Figure 4: (A) Parameter estimates from **penalized** (thin lines) and **glmnet** (dots) superimposed. (B) The difference between the parameter estimates. The jags are due to the slightly different convergence criteria in **glmnet** and **penalized**.

ciple." In B Petrov, F Csàki (eds.), *2nd International Symposium on Information Theory*. Akademiai Kiàdo, Budapest.

Andrew G, Gao J (2007). "Scalable Training of L1-regularized Log-linear Models." In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pp. 33–40. ACM, New York, NY, USA. ISBN 978-1-59593-793-3. doi:10.1145/1273496.1273501. URL http://doi.acm.org/10.1145/1273496.1273501.

Antoniadis A, Fan J (2001). "Regularization of Wavelet Approximations." *Journal of the American Statistical Association*, **96**(455), 939–967. ISSN 0162-1459. doi:10.1198/016214501753208942. URL http://amstat.tandfonline.com/doi/abs/10.1198/016214501753208942.

Fan J, Li R (2001). "Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties." *Journal of the American Statistical Association*, **96**(456), 1348–1360. ISSN 0162-1459. doi:10.1198/016214501753382273.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. ISSN 1548-7660. URL http://www.jstatsoft.org/v33/i01/paper/.

Hoerl AE, Kennard RW (1970). "Ridge Regression: Biased Estimation for Nonorthogonal Problems." *Technometrics*, **12**(1), 55–67. ISSN 0040-1706. doi:10.1080/00401706.1970.10488634. URL http://amstat.tandfonline.com/doi/abs/10.1080/00401706.1970.10488634.

Mallows CL (1973). "Some Comments on C_p." *Technometrics*, **15**(4), 661–675. ISSN 0040-1706. doi:10.1080/00401706.1973.10489103.

Mazumder R, Friedman JH, Hastie T (2011). "SparseNet: Coordinate Descent With Nonconvex Penalties." *Journal of the American Statistical Association*, **106**(495), 1125–1138. ISSN 0162-1459. doi:10.1198/jasa.2011.tm09738.

McCullagh P, Nelder JA (1989). *Generalized Linear Models, Second Edition*. Taylor and Francis. ISBN 9780412317606.

Meinshausen N (2007). "Relaxed Lasso." *Computational Statistics & Data Analysis*, **52**(1), 374–393. ISSN 0167-9473. doi:10.1016/j.csda.2006.12.019. URL http://www.sciencedirect.com/science/article/pii/S0167947306004956.

Nelder JA, Wedderburn RWM (1972). "Generalized Linear Models." *Journal of the Royal Statistical Society A*, **135**(3), 370–384. ISSN 00359238. doi:10.2307/2344614. URL http://www.jstor.org/stable/2344614.

Nikolova M (2000). "Local Strong Homogeneity of a Regularized Estimator." *SIAM Journal on Applied Mathematics*, **61**(2), 633–658. ISSN 0036-1399. doi:10.1137/S0036139997327794. URL http://epubs.siam.org/doi/abs/10.1137/S0036139997327794.

Park MY, Hastie T (2007). "L1-Regularization Path Algorithm for Generalized Linear Models." *Journal of the Royal Statistical Society B (Statistical Methodology)*, **69**(4), 659–677. ISSN 1467-9868. doi:10.1111/j.1467-9868.2007.00607.x.

Perkins S, Lacker K, Theiler J (2003). "Grafting: Fast, Incremental Feature Selection by Gradient Descent in Function Space." *J. Mach. Learn. Res.*, **3**, 1333–1356. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=944919.944976.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Radchenko P, James GM (2011). "Improved Variable Selection with Forward-Lasso Adaptive Shrinkage." *The Annals of Applied Statistics*, **5**(1), 427–448. ISSN 1932-6157, 1941-7330. doi:10.1214/10-AOAS375. URL http://projecteuclid.org/euclid.aoas/1300715197.

Schmidt M, Fung G, Rosales R (2009). "Optimization Methods for L1-Regularization." *Technical report*, University of British Columbia. Technical Report TR-2009-19.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**(2), 461–464. ISSN 0090-5364, 2168-8966. doi:10.1214/aos/1176344136. URL http://projecteuclid.org/euclid.aos/1176344136.

The MathWorks, Inc (2007). *MATLAB Release 2007b*. The MathWorks, Inc., Natick, Massachusetts, United States.

Tibshirani R (1996). "Regression Shrinkage and Selection via the LASSO." *Journal of the Royal Statistical Society B. Methodological*, **58**(1), 267–288. ISSN 0035-9246.

Zhang CH (2010). "Nearly Unbiased Variable Selection under Minimax Concave Penalty." *The Annals of Statistics*, **38**(2), 894–942. ISSN 0090-5364, 2168-8966. doi:10.1214/09-AOS729. URL http://projecteuclid.org/euclid.aos/1266586618.

Zou H (2006). "The Adaptive Lasso and Its Oracle Properties." *Journal of the American Statistical Association*, **101**(476), 1418–1429. ISSN 0162-1459. doi:10.1198/016214506000000735.

Zou H, Hastie T (2005). "Regularization and Variable Selection via the Elastic Net." *Journal of the Royal Statistical Society B*, **67**(2), 301–320. ISSN 1467-9868. doi:10.1111/j.1467-9868.2005.00503.x. URL http://onlinelibrary.wiley.com/doi/10.1111/j.1467-9868.2005.00503.x/abstract.

Zou H, Hastie T, Tibshirani R (2007). "On the degrees of freedom of the lasso." *The Annals of Statistics*, **35**(5), 2173–2192. ISSN 0090-5364, 2168-8966. doi:10.1214/009053607000000127. URL http://projecteuclid.org/euclid.aos/1194461726.

A. The FLASH penalty

The Forward Lasso Adaptive SHrinkage (FLASH) algorithm was described in Radchenko and James (2011). They did not, however, specify the objective function that the algorithm maximizes. Here it is shown that the FLASH algorithm is a form of penalized likelihood. We only consider the least-squares case.

At iteration t, the FLASH algorithm computes two updates. The first update is the ordinary least-squares update over an active set A:

$$\boldsymbol{\beta}_A^{t+1} = \boldsymbol{\beta}_A^t + (\mathbf{X}'_A\mathbf{X}_A)^{-1}\mathbf{X}'_A(\mathbf{y} - \mathbf{X}_A\boldsymbol{\beta}_A^t) \tag{14}$$

and the second is the lasso update

$$\boldsymbol{\beta}_A^{t+1} = \boldsymbol{\beta}_A^t + (\mathbf{X}'_A\mathbf{X}_A)^{-1}\mathbf{X}'_A(\mathbf{y} - \mathbf{X}_A\boldsymbol{\beta}_A^t - \lambda\text{sign}(\boldsymbol{\beta}_A^t)) \tag{15}$$

In either case, the step is $\boldsymbol{\beta}_A^{t+1} - \boldsymbol{\beta}_A^t$. Thus

$$\begin{aligned}\text{step}_{OLS} &= (\mathbf{X}'_A\mathbf{X}_A)^{-1}\mathbf{X}'_A(\mathbf{y} - \mathbf{X}_A\boldsymbol{\beta}_A^t)\\ \text{step}_{LASSO} &= (\mathbf{X}'_A\mathbf{X}_A)^{-1}\mathbf{X}'_A(\mathbf{y} - \mathbf{X}_A\boldsymbol{\beta}_A^t - \lambda\text{sign}(\boldsymbol{\beta}_A^t))\end{aligned}$$

FLASH takes a step which is a weighted sum of the least squares step and the LASSO step. That is, the FLASH update is

$$\begin{aligned}\boldsymbol{\beta}_A^{t+1} &= \boldsymbol{\beta}_A^t + \delta\,\text{step}_{OLS} + (1-\delta)\text{step}_{LASSO}\\ &= \boldsymbol{\beta}_A^t + (\mathbf{X}'_A\mathbf{X}_A)^{-1}\mathbf{X}'_A(\mathbf{y} - \mathbf{X}_A\boldsymbol{\beta}_A^t - (1-\delta)\lambda\text{sign}(\boldsymbol{\beta}_A^t))\end{aligned}$$

This implies that the penalty is $\pi(\beta) = (1-\delta)|\beta|$, for coefficients in the active set, which is the FLASH penalty defined here.

B. The multinomial model

This appendix describes how the multinomial model can be reformulated to fit the constraints of the maximization routine. The core idea behind this reformulation is that a single multinomial observation $(y_1, y_2, \ldots, y_q)$ can be thought of as q independent categorical observations $(y_1, 0, \ldots, 0), (0, y_2, \ldots, 0), \ldots, (0, 0, \ldots, y_q)$.

A set of multinomial observations is a matrix $\mathbf{Y}$ where the i,j-th element $y_{i,j}$ is the number of times category j occurred in observation i. Each element $y_{i,j}$ has a certain probability $p_{i,j}$ of occurring, and the log-likelihood of the observations is

$$L = \sum_{i=1}^{n} \sum_{j=1}^{q} y_{i,j} \log p_{i,j}$$

The matrix of observations $\mathbf{Y}$ can be viewed as a concatenated set of column vectors $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_q]$, one column for each category. Each column vector $\mathbf{y}_j$ has an associated probability vector

$$\mathbf{p}_j = \begin{bmatrix} p_{1,j} \\ p_{2,j} \\ \vdots \\ p_{n,j} \end{bmatrix}$$

The observation vectors $\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_q$ can be stacked into a single vector

$$\mathbf{y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_q \end{bmatrix}$$

and these observations are independent. The probability vectors $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_q$ can likewise be stacked to form a single vector

$$\mathbf{p} = \begin{bmatrix} \mathbf{p}_1 \\ \mathbf{p}_2 \\ \vdots \\ \mathbf{p}_q \end{bmatrix}$$

The log-likelihood can be written $L = \mathbf{y}^\top \log \mathbf{p}$, and is unchanged. Next we assume that each probability vector $\mathbf{p}_k$ can be written as

$$\mathbf{p}_k = \frac{exp(\boldsymbol{\eta}_{k,k})}{\sum_{j=1}^{q} exp(\boldsymbol{\eta}_{k,j})}$$

where each vector $\boldsymbol{\eta}_{k,j} = \tilde{\mathbf{X}}\boldsymbol{\beta}_j$ and the exponentiation and division are both element by element. $\tilde{\mathbf{X}}$ is a matrix of covariates. The coefficient vectors $\boldsymbol{\beta}_j$ can also be stacked to form a single coefficient matrix

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1 \\ \boldsymbol{\beta}_2 \\ \vdots \\ \boldsymbol{\beta}_q \end{bmatrix}$$

We wish to work out the gradient of L with respect to $\boldsymbol{\beta}$, which will be written $dL/d\boldsymbol{\beta}$. By the chain rule, this is

$$\frac{dL}{d\boldsymbol{\beta}} = \frac{d\mathbf{p}}{d\boldsymbol{\beta}} \frac{dL}{d\mathbf{p}}$$

The gradient $dL/d\mathbf{p}$ is simply $\mathbf{y}/\mathbf{p}$, where the division is element-by-element. The Jacobian $d\mathbf{p}/d\boldsymbol{\beta}$ is

$$\frac{d\mathbf{p}}{d\boldsymbol{\beta}} = \begin{bmatrix} d\mathbf{p}_1/d\boldsymbol{\beta}_1 & d\mathbf{p}_2/d\boldsymbol{\beta}_1 & \dots & d\mathbf{p}_q/d\boldsymbol{\beta}_1 \\ d\mathbf{p}_1/d\boldsymbol{\beta}_2 & d\mathbf{p}_2/d\boldsymbol{\beta}_2 & \dots & d\mathbf{p}_q/d\boldsymbol{\beta}_2 \\ \vdots & & \ddots & \\ d\mathbf{p}_1/d\boldsymbol{\beta}_q & d\mathbf{p}_2/d\boldsymbol{\beta}_q & \dots & d\mathbf{p}_q/d\boldsymbol{\beta}_q \end{bmatrix}$$

Each of the submatrices $d\mathbf{p}_k/d\boldsymbol{\beta}_j$ is itself a Jacobian matrix given by

$$\frac{d\mathbf{p}_k}{d\boldsymbol{\beta}_j} = \frac{d\boldsymbol{\eta}_{k,1}}{d\boldsymbol{\beta}_j}\frac{d\mathbf{p}_k}{d\boldsymbol{\eta}_{k,1}} + \frac{d\boldsymbol{\eta}_{k,2}}{d\boldsymbol{\beta}_j}\frac{d\mathbf{p}_k}{d\boldsymbol{\eta}_{k,2}} + \ldots + \frac{d\boldsymbol{\eta}_{k,q}}{d\boldsymbol{\beta}_j}\frac{d\mathbf{p}_k}{d\boldsymbol{\eta}_{k,q}} = X^\top \frac{d\mathbf{p}_k}{d\boldsymbol{\eta}_{k,j}}$$

where $d\mathbf{p}_k/d\boldsymbol{\eta}_{k,j}$ is a diagonal matrix of derivatives.

B.1. Interface to the maximization algorithm

The scoring routine requires a tuple $\{\mathbf{m}, \mathbf{D}, \mathbf{V}, \mathbf{J}\}$. The vector $\mathbf{m}$ is just

$$\mathbf{m} = \frac{dL}{d\mathbf{p}} = \frac{\mathbf{y}}{\mathbf{p}}$$

where division is element-by-element. The vector $\mathbf{V}$ is the expected value of $\mathbf{m}^2$, namely

$$\mathbf{V} = E\left(\frac{\mathbf{y}}{\mathbf{p}}\right)^2 = \mathbf{y}^2 - \mathbf{y} + \frac{\mathbf{y}}{\mathbf{p}} = \mathbf{y}^2 - \mathbf{y} + \mathbf{m}$$

Unfortunately, $\mathbf{D}$ and $\mathbf{X}$ are not so simple. The product $\mathbf{DX}$ must equal the Jacobian $d\mathbf{p}/d\boldsymbol{\beta}$ The best value for $\mathbf{D}$ is 1, and $\mathbf{X}$ is the matrix

$$\begin{aligned} \mathbf{X} = \left(\frac{d\mathbf{p}}{d\boldsymbol{\beta}}\right)^\top &= \begin{bmatrix} d\mathbf{p}_1/d\boldsymbol{\beta}_1 & d\mathbf{p}_1/d\boldsymbol{\beta}_2 & \dots & d\mathbf{p}_1/d\boldsymbol{\beta}_q \\ d\mathbf{p}_2/d\boldsymbol{\beta}_1 & d\mathbf{p}_2/d\boldsymbol{\beta}_2 & \dots & d\mathbf{p}_2/d\boldsymbol{\beta}_q \\ \vdots & & \ddots & \\ d\mathbf{p}_q/d\boldsymbol{\beta}_1 & d\mathbf{p}_q/d\boldsymbol{\beta}_2 & \dots & d\mathbf{p}_q/d\boldsymbol{\beta}_q \end{bmatrix} \\ &= \begin{bmatrix} D_{1,1}\tilde{\mathbf{X}} & D_{1,2}\tilde{\mathbf{X}} & \dots & D_{1,q}\tilde{\mathbf{X}} \\ D_{2,1}\tilde{\mathbf{X}} & D_{2,2}\tilde{\mathbf{X}} & \dots & D_{2,q}\tilde{\mathbf{X}} \\ \vdots & & \ddots & \\ D_{q,1}\tilde{\mathbf{X}} & D_{q,2}\tilde{\mathbf{X}} & \dots & D_{q,q}\tilde{\mathbf{X}} \end{bmatrix} \end{aligned}$$

in which $D_{i,j} = d\mathbf{p_i}/d\boldsymbol{\eta}_j$ is a diagonal matrix and $\tilde{\mathbf{X}}$ is the matrix of covariates mentioned earlier.

Affiliation:

William McIlhagga
Bradford School of Optometry and Vision Science
University of Bradford
Bradford BD7 1DP, England
E-mail: w.h.mcilhagga@bradford.ac.uk