SDS 385: Homework 6

G. Paulon

October 17, 2016

Problem 1. Proximal operators

Let $f(x)$ be a convex function. The **Moreau envelope** $E_\gamma f(x)$ and **proximal operator** $\text{prox}_\gamma f(x)$ for parameter $\gamma > 0$ are defined as

$$E_\gamma f(x) = \min_z \left\{ f(z) + \frac{1}{2\gamma}||z - x||_2^2 \right\}$$
$$\text{prox}_\gamma f(x) = \underset{z}{\text{argmin}} \left\{ f(z) + \frac{1}{2\gamma}||z - x||_2^2 \right\}.$$

The proximal operator of a function evaluated at one point moves the point towards the minimum. More precisely, the transformed point $\text{prox}_\gamma f(x)$ is a compromise between minimizing f and being near to x. The parameter γ controls this trade-off between these two terms. The Moreau envelope, instead, is a regularized version of f. It approximates f from below, and has the same set of minimizing values as f.

(A) *The proximal operator gives a nice interpretation of classical gradient descent. Consider the local linear approximation of $f(\boldsymbol{x})$ about a point $\boldsymbol{x}_0$:*

$$f(\boldsymbol{x}) \approx \hat{f}(\boldsymbol{x}; \boldsymbol{x}_0) = f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla f(\boldsymbol{x}_0).$$

Derive the proximal operator (with parameter γ) of the linear approximation $\hat{f}(\boldsymbol{x}; \boldsymbol{x}_0)$, and show that this proximal operator is identical to a gradient descent step for $f(\boldsymbol{x})$ of size γ, starting from the point $\boldsymbol{x}_0$.

Computing the proximal operator of $\hat{f}(\boldsymbol{x}; \boldsymbol{x}_0)$ is equivalent to moving towards the minimum of $\hat{f}(\boldsymbol{x}; \boldsymbol{x}_0)$ staying still close to $\boldsymbol{x}_0$. This makes sense, since the problem consists in minimizing a **local** linear approximation and so it requires to stay close to $\boldsymbol{x}_0$.

$$\text{prox}_\gamma \hat{f}(\boldsymbol{x}; \boldsymbol{x}_0) = \underset{\boldsymbol{x}}{\text{argmin}} \left\{ f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla f(\boldsymbol{x}_0) + \frac{1}{2\gamma}||\boldsymbol{x} - \boldsymbol{x}_0||_2^2 \right\}$$

This problem can be easily solved by computing the gradient of the objective function

$$\begin{aligned} &\nabla_{\boldsymbol{x}} \left\{ f(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla f(\boldsymbol{x}_0) + \frac{1}{2\gamma}(\boldsymbol{x} - \boldsymbol{x}_0)^T (\boldsymbol{x} - \boldsymbol{x}_0) \right\} \\ =&\nabla f(\boldsymbol{x}_0) + \frac{1}{\gamma}(\boldsymbol{x} - \boldsymbol{x}_0) \end{aligned}$$

which is then set to 0, yielding

$$\hat{\boldsymbol{x}} = \text{prox}_\gamma \hat{f}(\boldsymbol{x}; \boldsymbol{x}_0) = \boldsymbol{x}_0 - \gamma \nabla f(\boldsymbol{x}_0),$$

which is exactly the gradient update. Therefore, the gradient step minimizes a local linear approximation of the function in a neighbourhood of the current iterate (where, presumably, the linear approximation is reasonable).

(B) *Many intermediate steps in statistical optimization problems can be written very compactly in terms of proximal operators of log-likelihoods or penalty functions. For example, consider a negative log likelihood of the form*

$$l(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^T P\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} + r.$$

First, show that if we have a Gaussian sampling model of the form $(\boldsymbol{y}|\boldsymbol{x}) \sim N(A\boldsymbol{x}, \Omega^{-1})$, then our negative log-likelihood can be written in the form given above. Then show that the proximal operator with parameter $1/\gamma$ of $l(\boldsymbol{x})$ takes the form

$$prox_{\frac{1}{\gamma}} l(\boldsymbol{x}) = (P + \gamma I)^{-1}(\gamma\boldsymbol{x} - \boldsymbol{q})$$

assuming the relevant inverse exists.

This is a multivariate normal model, whose likelihood function can be written as

$$L(\boldsymbol{y}; \boldsymbol{x}) \propto \exp\left\{-\frac{1}{2}(\boldsymbol{y} - A\boldsymbol{x})^T\Omega(\boldsymbol{y} - A\boldsymbol{x})\right\}.$$

Therefore the negative log-likelihood is

$$\begin{aligned} l(\boldsymbol{y}; \boldsymbol{x}) &\propto \frac{1}{2}(\boldsymbol{y} - A\boldsymbol{x})^T\Omega(\boldsymbol{y} - A\boldsymbol{x}) \\ &= \frac{1}{2}\boldsymbol{y}^T\Omega\boldsymbol{y} + \frac{1}{2}\boldsymbol{x}^T A^T\Omega A\boldsymbol{x} - \boldsymbol{y}^T\Omega A\boldsymbol{x} \\ &= \frac{1}{2}\boldsymbol{x}^T P\boldsymbol{x} + \boldsymbol{q}^T\boldsymbol{x} + r \end{aligned}$$

where

$$\begin{aligned} P &= A^T\Omega A \\ \boldsymbol{q} &= -A^T\Omega\boldsymbol{y} \\ r &= \frac{1}{2}\boldsymbol{y}^T\Omega\boldsymbol{y}. \end{aligned}$$

The proximal operator is then

$$\begin{aligned} \text{prox}_{\frac{1}{\gamma}} l(\boldsymbol{x}) &= \underset{\boldsymbol{z}}{\text{argmin}}\left\{l(\boldsymbol{z}) + \frac{\gamma}{2}||\boldsymbol{z} - \boldsymbol{x}||_2^2\right\} \\ &= \underset{\boldsymbol{z}}{\text{argmin}}\left\{\frac{1}{2}\boldsymbol{z}^T P\boldsymbol{z} + \boldsymbol{q}^T\boldsymbol{z} + r + \frac{\gamma}{2}(\boldsymbol{z} - \boldsymbol{x})^T(\boldsymbol{z} - \boldsymbol{x})\right\}. \end{aligned}$$

The minimum can be found by computing the gradient

$$\begin{aligned} &\nabla\left\{\frac{1}{2}\boldsymbol{z}^T P\boldsymbol{z} + \boldsymbol{q}^T\boldsymbol{z} + r + \frac{\gamma}{2}(\boldsymbol{z} - \boldsymbol{x})^T(\boldsymbol{z} - \boldsymbol{x})\right\} \\ =&P\boldsymbol{z} + \boldsymbol{q} + \frac{\gamma}{2}2(\boldsymbol{z} - \boldsymbol{x}), \end{aligned}$$

where in the last equality we exploited the fact that P is a symmetric matrix. Therefore the optimum is obtained by setting the gradient equal to 0, that is

$$\begin{aligned} &(P + \gamma I)\hat{\boldsymbol{z}} = \gamma\boldsymbol{x} - \boldsymbol{q} \\ \Rightarrow&\hat{\boldsymbol{z}} = \text{prox}_{\frac{1}{\gamma}} l(\boldsymbol{x}) = (P + \gamma I)^{-1}(\gamma\boldsymbol{x} - \boldsymbol{q}) \end{aligned}$$

(C) *Let $\phi(\boldsymbol{x}) = \tau||\boldsymbol{x}||_1$. Express the proximal operator of this function in terms of the soft-thresholding function that we learned about in the last set of exercises.*

The proximal operator of this function is

$$\text{prox}_\gamma \phi(\boldsymbol{x}) = \underset{\boldsymbol{z}}{\text{argmin}} \left\{ \tau||\boldsymbol{z}||_1 + \frac{1}{2\gamma}||\boldsymbol{z} - \boldsymbol{x}||_2^2 \right\}.$$

To solve this minimization problem, as usual, we compute the gradient and we set it to 0. Let us compute the generic partial derivative (element-wise expression):

$$\begin{aligned} &\frac{\partial}{\partial z_i} \left\{ \tau|z_i| + \frac{1}{2\gamma}(z_i - x_i)^2 \right\} \Big|_{z_i = \hat{z}_i} \\ =&\tau \text{sign}(\hat{z}_i) + \frac{1}{\gamma}(\hat{z}_i - x_i). \end{aligned}$$

Let us now split the three cases:

- if $\hat{z}_i > 0$, then $\hat{z}_i = x_i - \tau\gamma$ under the constraint $x_i > \tau\gamma$;
- if $\hat{z}_i < 0$, then $\hat{z}_i = x_i + \tau\gamma$ under the constraint $x_i < -\tau\gamma$;
- if $\hat{z}_i = 0$, that means that $-\tau\gamma < x_i < \tau\gamma$.

Therefore, a compact form can be written as

$$\hat{\boldsymbol{z}} = \text{prox}_\gamma \phi(\boldsymbol{x}) = \text{sign}(\boldsymbol{x})(|\boldsymbol{x}| - \tau\gamma \mathbf{1})_+ = S_{\tau\gamma}(\boldsymbol{x}).$$

Problem 2. The proximal gradient method

Suppose that we have some objective function that can be expressed as $f(\boldsymbol{x}) = l(\boldsymbol{x}) + \phi(\boldsymbol{x})$, where $l(\boldsymbol{x})$ is differentiable but $\phi(\boldsymbol{x})$ is not. Recall from above the idea of forming a local linear approximation to a function at some point $\boldsymbol{x}_0$ and then adding a quadratic regularizer. This gave us an interpretation of gradient descent evaluating the proximal operator of our locally linear approximation. Here, we will apply this idea to the first term in our objective, $l(\boldsymbol{x})$. Define

$$l(\boldsymbol{x}) \approx \tilde{l}(\boldsymbol{x}; \boldsymbol{x}_0) = l(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla l(\boldsymbol{x}_0) + \frac{1}{2\gamma}||\boldsymbol{x} - \boldsymbol{x}_0||_2^2$$

as our linear approximation to $l(\boldsymbol{x})$, plus the quadratic regularizer. Now we add in the $\phi(\boldsymbol{x})$ term to get the approximation for our original objective:

$$f(\boldsymbol{x}) \approx \tilde{f}(\boldsymbol{x}; \boldsymbol{x}_0) = \tilde{l}(\boldsymbol{x}; \boldsymbol{x}_0) + \phi(\boldsymbol{x}).$$

(A) *Consider the surrogate optimization problem in which we minimize the approximation $\tilde{f}(\boldsymbol{x}; \boldsymbol{x}_0)$ in lieu of of our original objective $f(\boldsymbol{x})$, i.e.*

$$\hat{\boldsymbol{x}} = \underset{\boldsymbol{x}}{\text{argmin}} \left\{ \tilde{l}(\boldsymbol{x}; \boldsymbol{x}_0) + \phi(\boldsymbol{x}) \right\}.$$

Show that the solution to this problem is of the form

$$\hat{\boldsymbol{x}} = \mathit{prox}_{\gamma}\phi(\boldsymbol{u}), \quad \mathit{where} \quad \boldsymbol{u} = \boldsymbol{x}_0 - \gamma\nabla l(\boldsymbol{x}_0).$$

This is just the proximal operator of the non-smooth part of the objective, $\phi(\boldsymbol{x})$, evaluated at an intermediate gradient-descent step for the smooth part, $l(\boldsymbol{x})$.

We have to solve the following minimization problem:

$$\begin{aligned}\hat{\boldsymbol{x}} &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \tilde{l}(\boldsymbol{x}; \boldsymbol{x}_0) + \phi(\boldsymbol{x}). \right\} \\ &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ l(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla l(\boldsymbol{x}_0) + \frac{1}{2\gamma}||\boldsymbol{x} - \boldsymbol{x}_0||_2^2 + \phi(\boldsymbol{x}) \right\}.\end{aligned}$$

Since the first term of the sum does not depend on $\boldsymbol{x}$, we can get rid of it. For the same reason, we can introduce another term (its purpose will be clear at the end of the proof) $\frac{\gamma}{2}\nabla l(\boldsymbol{x}_0)^T \nabla l(\boldsymbol{x}_0)$ which does not depend on $\boldsymbol{x}$, and the minimization problem is the same. Therefore

$$\begin{aligned}\hat{\boldsymbol{x}} &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ l(\boldsymbol{x}_0) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla l(\boldsymbol{x}_0) + \frac{1}{2\gamma}||\boldsymbol{x} - \boldsymbol{x}_0||_2^2 + \phi(\boldsymbol{x}) \right\} \\ &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \phi(\boldsymbol{x}) + (\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla l(\boldsymbol{x}_0) + \frac{1}{2\gamma}(\boldsymbol{x} - \boldsymbol{x}_0)^T(\boldsymbol{x} - \boldsymbol{x}_0) + \frac{\gamma}{2}\nabla l(\boldsymbol{x}_0)^T \nabla l(\boldsymbol{x}_0) \right\} \\ &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \phi(\boldsymbol{x}) + \frac{1}{2\gamma}\left[(\boldsymbol{x} - \boldsymbol{x}_0)^T(\boldsymbol{x} - \boldsymbol{x}_0) + 2\gamma(\boldsymbol{x} - \boldsymbol{x}_0)^T \nabla l(\boldsymbol{x}_0) + \gamma^2 \nabla l(\boldsymbol{x}_0)^T \nabla l(\boldsymbol{x}_0) \right] \right\} \\ &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \phi(\boldsymbol{x}) + \frac{1}{2\gamma}(\boldsymbol{x} - \boldsymbol{x}_0 + \gamma\nabla l(\boldsymbol{x}_0))^T(\boldsymbol{x} - \boldsymbol{x}_0 + \gamma\nabla l(\boldsymbol{x}_0)) \right\} \\ &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \phi(\boldsymbol{x}) + \frac{1}{2\gamma}||\boldsymbol{x} - \boldsymbol{x}_0 + \gamma\nabla l(\boldsymbol{x}_0)||_2^2 \right\} \\ &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \phi(\boldsymbol{x}) + \frac{1}{2\gamma}||\boldsymbol{x} - (\boldsymbol{x}_0 - \gamma\nabla l(\boldsymbol{x}_0))||_2^2 \right\}\end{aligned}$$

Let us now define $\boldsymbol{u} = \boldsymbol{x}_0 - \gamma\nabla l(\boldsymbol{x}_0)$. Then,

$$\begin{aligned}\hat{\boldsymbol{x}} &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \phi(\boldsymbol{x}) + \frac{1}{2\gamma}||\boldsymbol{x} - (\boldsymbol{x}_0 - \gamma\nabla l(\boldsymbol{x}_0))||_2^2 \right\} \\ &= \underset{\boldsymbol{x}}{\operatorname{argmin}} \left\{ \phi(\boldsymbol{x}) + \frac{1}{2\gamma}||\boldsymbol{x} - \boldsymbol{u}||_2^2 \right\} \\ &= \operatorname{prox}_{\gamma}\phi(\boldsymbol{u}).\end{aligned}$$

(B) *The proximal gradient method is an iterative algorithm that consists in the following step*

$$\boldsymbol{x}^{(t+1)} = \mathit{prox}_{\gamma^{(t)}}\phi(\boldsymbol{u}^{(t)}), \quad \boldsymbol{u}^{(t)} = \boldsymbol{x}^{(t)} - \gamma^{(t)}\nabla l(\boldsymbol{x}^{(t)}).$$

Now consider the lasso regression problem:

$$\hat{\boldsymbol{\beta}} = \underset{\boldsymbol{\beta}}{\operatorname{argmin}} \left\{ \frac{1}{2}||\boldsymbol{y} - X\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_1 \right\}.$$

Using the results on proximal operators you have derived already, write down some concise pseudo-code for using the proximal gradient algorithm to minimize this objective. Identify the primary computational costs of this algorithm.

In the Lasso context, we can set

$$l(\boldsymbol{\beta}) = \frac{1}{2}||\boldsymbol{y} - X\boldsymbol{\beta}||_2^2, \quad \phi(\boldsymbol{\beta}) = \lambda||\boldsymbol{\beta}||_1,$$

where the first function in differentiable and the second is not. The proximal gradient algorithm consists then in the minimization of the approximation of the objective $\tilde{l}(\boldsymbol{\beta}; \boldsymbol{\beta}_0) + \phi(\boldsymbol{\beta})$. The update step takes the following form

$$\boldsymbol{\beta}^{(t+1)} = \text{prox}_{\gamma^{(t)}\phi}(\boldsymbol{u}^{(t)}) = S_{\gamma^{(t)}\lambda}(\boldsymbol{u}^{(t)}),$$

where $\boldsymbol{u}^{(t)} = \boldsymbol{\beta}^{(t)} - \gamma^{(t)}\nabla l(\boldsymbol{\beta}^{(t)})$. Let us recall that

$$\begin{aligned}\nabla l(\boldsymbol{\beta}) &= \nabla \left\{ \frac{1}{2}(\boldsymbol{y} - X\boldsymbol{\beta})^T(\boldsymbol{y} - X\boldsymbol{\beta}) \right\} \\ &= -X^T(\boldsymbol{y} - X\boldsymbol{\beta}),\end{aligned}$$

and therefore the update step becomes

$$\boldsymbol{u}^{(t)} = \boldsymbol{\beta}^{(t)} + \gamma^{(t)}X^T(\boldsymbol{y} - X\boldsymbol{\beta}^{(t)}).$$

Evaluating $\nabla l(\boldsymbol{\beta})$ requires two matrix-vector multiplications, plus a negligible vector addition. Evaluating the proximal operator of $\phi(\boldsymbol{u})$ is negligible.

Algorithm 1 Proximal Gradient

```
procedure PROXIMAL.GRADIENT(y, X, β^(1), λ, γ = 10^-4)
    for iter = 2:maxiter do
        gradient ← GRAD.LOGLIK(β^(iter−1), y, X)
        u^(t) ← β^(iter−1) − γ · gradient
        β^(iter) ← SOFT.THRESHOLDING(u^(t), γλ)
```

(C) *A cool variation on proximal gradient is called the accelerated proximal gradient algorithm. The following scheme involves a simple extrapolation step based on the previous iteration:*

$$\begin{aligned}&\boldsymbol{\beta}^{(t+1)} = \text{prox}_{\gamma^{(t)}\phi}(\boldsymbol{u}^{(t)}), \quad \boldsymbol{u}^{(t)} = \boldsymbol{z}^{(t)} - \gamma^{(t)}\nabla l(\boldsymbol{z}^{(t)}) \\ &s^{(t+1)} = \frac{1 + (1 + 4(s^{(t)})^2)^{1/2}}{2} \\ &\boldsymbol{z}^{(t+1)} = \boldsymbol{\beta}^{(t+1)} + \left(\frac{s^{(t)} - 1}{s^{(t+1)}}\right)(\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\beta}^{(t)})\end{aligned}$$

The first step involves the proximal operator of the penalty function, evaluated not at the previous iterate $\boldsymbol{\beta}^{(t)}$, but at an extrapolated version of $\boldsymbol{\beta}^{(t)}$, based on the magnitude of the previous step's

update ($\boldsymbol{\beta}^{(t+1)} - \boldsymbol{\beta}^{(t)}$). In this sense, large steps give "momentum" to the next step, where the amount of momentum is modulated by the scalar $s^{(t)}$ terms. Implement this acceleration scheme in your proximal gradient code, and compare its convergence speed to that of the unacceleration version. Does the value of the objective goes down at each and every step? Do you get to the minimum faster?

Algorithm 2 Accelerated Proximal Gradient

procedure ACC.PROXIMAL.GRADIENT($y, X, \boldsymbol{\beta}^{(1)}, \lambda, \gamma = 10^{-4}$)
  for iter = 2:maxiter **do**
    $gradient \leftarrow$ GRAD.LOGLIK($\boldsymbol{z}^{(iter-1)}, y, X$)
    $\boldsymbol{u}^{(t)} \leftarrow \boldsymbol{z}^{(iter-1)} - \gamma \cdot gradient$
    $\boldsymbol{\beta}^{(iter)} \leftarrow$ SOFT.THRESHOLDING($\boldsymbol{u}^{(t)}, \gamma\lambda$)
    $s^{(iter)} \leftarrow \frac{1+\sqrt{1+4(s^{(iter-1)})^2}}{2}$
    $\boldsymbol{z}^{(iter)} \leftarrow \boldsymbol{\beta}^{(iter)} + (\frac{s^{(iter-1)}-1}{s^{(iter)}})(\boldsymbol{\beta}^{(iter)} - \boldsymbol{\beta}^{(iter-1)})$

In Figure 1 we see the performance of the two algorithms. We used, as a convergence criterion, the relative decrement of the objective function. The accelerated version converges faster than the normal one. The two different algorithms have then been compared to the results obtained by R's `glmnet` package, showing the exact same estimated $\boldsymbol{\beta}$ values.

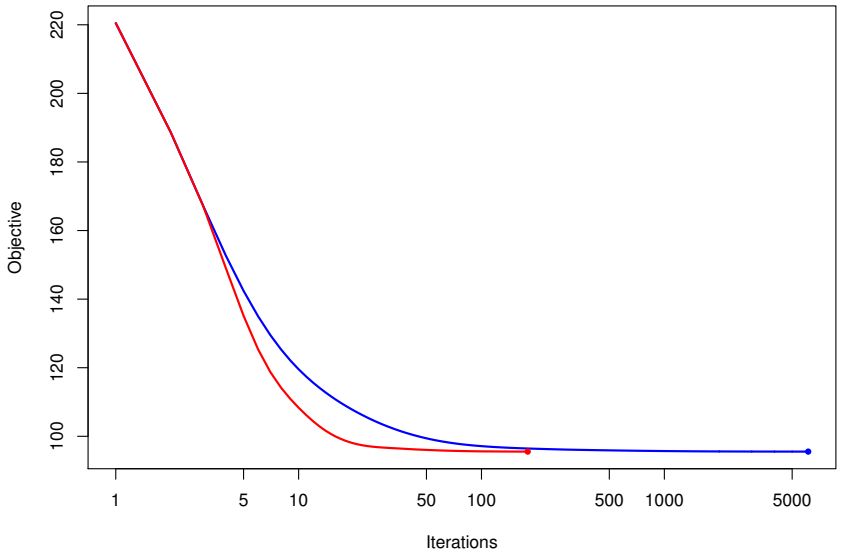

Figure 1: Convergence of the two algorithms. In red, the accelerated version of the proximal gradient.

Appendix A

R code

```r
log.lik <- function(beta, y, X, lambda){
  # Computes the objective function of the logit model
  # -------------------------------------------------
  # Args:
  #   - beta: regression parameters (length p)
  #   - y: response vector (length n)
  #   - X: matrix of the features (n*p)
  #   - lambda: penalization parameter
  # Returns:
  #   - ll: the scalar associated with the objective function at beta
  # -----------------------------------------------------------------
  Xbeta <- X %*% beta
  ll <- (1/2)*crossprod(y - Xbeta) + lambda * sum(abs(beta))
  return(ll)
}

grad.loglik <- function(beta, y, X){
  # Computes the gradient of negative log-likelihood of the logit model
  # -------------------------------------------------------------------
  # Args:
  #   - beta: regression parameters (length p)
  #   - y: response vector (length n)
  #   - X: matrix of the features (n*p)
  # Returns:
  #   - grad.ll: the gradient vector of the negative log-lik of the data (length p)
  # -------------------------------------------------------------------------------
  grad.ll <- array(NA, dim = length(beta))
  Xbeta <- X %*% beta
  grad.ll <- - crossprod(X, as.numeric(y - Xbeta))
  return(grad.ll)
}

soft.thresholding <- function(x, lambda){
  # Computes the soft thresholding estimator
  # ----------------------------------------
  # Args:
  #   - x: vector of the observations
  #   - lambda: penalization parameter (threshold)
  # Returns:
  #   - theta: the soft thresholding estimator
```

```
  # ----------------------------------------
  theta <- sign(x) * pmax(rep(0, length(x)), abs(x) - lambda)
  return (theta)
}
```

Listing A.1: Auxiliary functions.

```
proximal.gradient <- function(y, X, beta0, lambda, gamma = 1E-4, maxiter = 50000, tol = 1E-10){
  # Proximal gradient algorithm that allows to compute the Lasso regression estimates
  # ------------------------------------------------------------------------------------
  # Args:
  #   - y: response vector (length n)
  #   - X: matrix of the features (n*p)
  #   - beta0: initial values of beta
  #   - lambda: penalization parameter for the lasso (threshold)
  #   - gamma: proximal operator parameter
  #   - maxiter: maximum number of iterations
  #   - tol: tolerance defining convergence
  # Returns:
  #   - ll: values of the negative log-likelihood for each iteration
  #   - betas: values of the beta parameters for each iteration
  # ----------------------------------------------------------

  # Check if the inputs have the correct sizes
  if ((length(y) != dim(X)[1]) || (length(beta0) != dim(X)[2])){
    stop("Incorrect input dimensions.")
  }

  # Initialize the data structures
  betas <- array(NA, dim=c(maxiter, length(beta0)))
  betas[1,] <- beta0 # Initial guess
  ll <- array(NA, dim = maxiter)
  ll[1] <- log.lik(betas[1,], y, X, lambda)

  for (iter in 2:maxiter){

    # Compute the gradient
    gradient <- grad.loglik(betas[iter-1, ], y, X)
    u <- betas[iter-1, ] - gamma * gradient

    # Update beta
    betas[iter, ] <- soft.thresholding(u, gamma * lambda)

    # Compute log-likelihood
    ll[iter] <- log.lik(betas[iter,], y, X, lambda)

    # Convergence check
    if ( abs(ll[iter-1] - ll[iter]) / (ll[iter-1] + 1E-3) < tol ){
      cat('Algorithm has converged after', iter, 'iterations')
      ll <- ll[1:iter]
      betas <- betas[1:iter, ]
      break;
    }
    else if ( (iter == maxiter) & (abs(ll[iter-1] - ll[iter]) / (ll[iter-1] + 1E-3) >= tol) ){
      print('WARNING: algorithm has not converged')
      break;
    }
```

```
  }
  return(list("ll" = ll, "betas" = betas[iter,]))
}
```

Listing A.2: Proximal gradient algorithm.

```
acc.proximal.gradient <- function(y, X, beta0, lambda, gamma = 1E-4, maxiter = 50000, tol = 1E
    -10){
  # Accelerated proximal gradient algorithm that allows to compute the Lasso regression estimates
  # ------------------------------------------------------------------------------------------
  # Args:
  #   - y: response vector (length n)
  #   - X: matrix of the features (n*p)
  #   - beta0: initial values of beta
  #   - lambda: penalization parameter for the lasso (threshold)
  #   - gamma: proximal operator parameter
  #   - maxiter: maximum number of iterations
  #   - tol: tolerance defining convergence
  # Returns:
  #   - ll: values of the objective function for each iteration
  #   - betas: values of the beta parameters for each iteration
  # ----------------------------------------------------------

  # Check if the inputs have the correct sizes
  if ((length(y) != dim(X)[1]) || (length(beta0) != dim(X)[2])){
    stop("Incorrect input dimensions.")
  }

  # Initialize the data structures
  betas <- array(NA, dim=c(maxiter, length(beta0)))
  betas[1,] <- beta0 # Initial guess
  z <- array(NA, dim=c(maxiter, length(beta0)))
  z[1,] <- beta0
  s <- array(NA, dim = maxiter)
  s[1] <- 1
  ll <- array(NA, dim = maxiter)
  ll[1] <- log.lik(betas[1,], y, X, lambda)

  for (iter in 2:maxiter){

    # Compute the gradient
    gradient <- grad.loglik(z[iter-1, ], y, X)
    u <- z[iter-1, ] - gamma * gradient

    # Update beta
    betas[iter, ] <- soft.thresholding(u, gamma * lambda)
    s[iter] <- (1 + sqrt(1 + 4 * s[iter-1]^2))/2
    z[iter, ] <- betas[iter, ] + ((s[iter-1] - 1)/s[iter]) * (betas[iter, ] - betas[iter-1, ])

    # Compute log-likelihood
    ll[iter] <- log.lik(betas[iter,], y, X, lambda)

    # Convergence check
    if ( abs(ll[iter-1] - ll[iter]) / (ll[iter-1] + 1E-3) < tol ){
      cat('Algorithm has converged after', iter, 'iterations')
      ll <- ll[1:iter]
      betas <- betas[1:iter, ]
```

```
      break;
    }
    else if ( (iter == maxiter) & (abs(ll[iter-1] - ll[iter]) / (ll[iter-1] + 1E-3) >= tol) ){
      print('WARNING: algorithm has not converged')
      break;
    }
  }
  return(list("ll" = ll, "betas" = betas[iter,]))
}
```

Listing A.3: Accelerated proximal gradient algorithm.