Exercises 6 The Proximal Gradient Method

Natalia Zuniga-Garcia

October 23, 2017

In this set of exercises, we learn about the proximal gradient algorithm. This is our first all-purpose algorithm capable of handling non-smooth terms that enforce sparsity, like an ℓ^1 regularizer.

1 Proximal operators

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

$$E_\gamma f(x) = \min_z \left\{ f(z) + \frac{1}{2\gamma}\|z - x\|_2^2 \right\} \leq f(x) \tag{1}$$

$$\operatorname*{prox}_\gamma f(x) = \arg\min_z \left\{ f(z) + \frac{1}{2\gamma}\|z - x\|_2^2 \right\}. \tag{2}$$

Intuitively, the Moreau envelope is a regularized version of f. It approximates f from below, and has the same set of minimizing values as f. The proximal mapping returns the value that solves the little minimization problem defined by the Moreau envelope. The objective in this little minimization problem balances two goals: minimizing f, and staying near x. The proximal operator says *where* the minimum occurs, while the Moreau envelope says what the *value* of the minimum is.

Figure 1 shows a simple one-dimensional example of a Moreau envelope and a proximal operator for the absolute-value function.

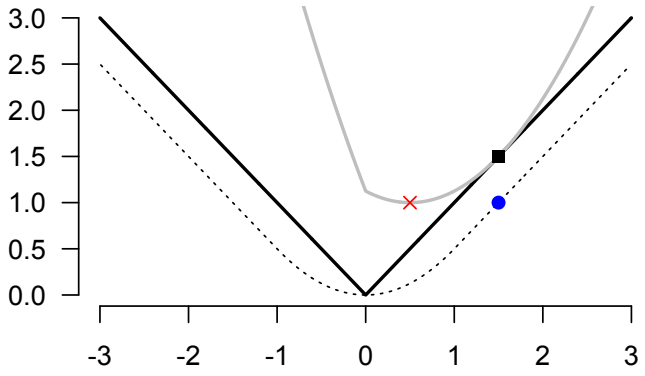

Figure 1: This picture shows a simple example of a proximal operator and Moreau envelope. The solid black line shows the function $f(x) = |x|$, and the dotted line shows the corresponding Moreau envelope $E_\gamma f(x)$ with parameter $\gamma = 1$. The grey line shows the function $|x| + (1/2)(x - x_0)^2$ for $x_0 = 1.5$, whose minimum (shown as a red cross) defines the Moreau envelope and proximal operator. This point has horizontal coordinate $\operatorname{prox}_\gamma f(x_0) = 0.5$ and vertical coordinate $E_\gamma f(x) = 1$, and is closer than x_0 to the overall minimum at $x = 0$. The blue circle shows the point $(x_0, E_\gamma f(x_0))$; the vertical coordinate of the blue point is precisely the vertical coordinate of the red point, emphasizing the point-wise construction of the Moreau envelope in terms of a simple optimization problem.

(A) The proximal operator gives a nice interpretation of classical gradient descent. Consider the local linear approximation of $f(x)$ about a point x_0:

$$f(x) \approx \hat{f}(x; x_0) = f(x_0) + (x - x_0)^{(t)} \nabla f(x_0)\,.$$

Derive the proximal operator (with parameter γ) of the linear approximation $\hat{f}(x; x_0)$, and show that this proximal operator is identical to a gradient-descent step for $f(x)$ of size γ, starting from the point x_0. In reflecting on this answer, make sure you feel comfortable with the following statement: "the gradient step minimizes a local linear approximation of the function, subject to a quadratic regularizer that keeps the next iterate close to x_0 (where, presumably, the linear approximation is reasonable)."

Solution

The proximal operator is:

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

Using the local linear approximation of $f(x)$ about a point x_0:

$$\underset{\gamma}{\text{prox}}\, \hat{f}(x; x_0) = \arg\min_z \left\{ f(x_0) + (z - x_0)^{(t)} \nabla f(x_0) + \frac{1}{2\gamma} \|z - x_0\|_2^2 \right\}$$

Now we try to minimize the objective function by calculation its derivative with respect to z and equalizing it to zero as follow,

$$\nabla f(x_0) + \frac{1}{\gamma}(z - x_0) = 0$$

$$z = x_0 - \gamma \nabla f(x_0)$$

We can observe that this is the gradient-descent step of size γ.

(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

$$l(x) = \frac{1}{2}x^T Px - q^T x + r\,.$$

Show that the proximal operator with parameter $1/\gamma$ of $l(x)$ takes the form

$$\underset{1/\gamma}{\text{prox}}\, l(x) = (P + \gamma I)^{-1}(\gamma x + q)\,,$$

assuming the relevant inverse exists.

Solution

The proximal operator with parameter $1/\gamma$ of $l(x)$ is:

$$\underset{1/\gamma}{\text{prox}}\, l(x) = \arg\min_z \left\{ l(z) + \frac{\gamma}{2}\|z - x\|_2^2 \right\} = \arg\min_z \left\{ \frac{1}{2}z^T Pz - q^T z + r + \frac{\gamma}{2}\|z - x\|_2^2 \right\}$$

Now we try to minimize by calculation its derivative with respect to z and equalizing it to zero as follow,

$$0 \overset{set}{=} Pz - q + \gamma(z - x)$$
$$-Pz - \gamma z = -q - \gamma x$$
$$(P + \gamma I)z = \gamma x + q$$

Where I is the identity matrix. Assuming that $(P + \gamma I)$ is invertible,

$$z = (P + \gamma I)^{-1}(\gamma x + q)$$

Then, we showed that,

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

Then show that if we have a Gaussian sampling model of the form $(y \mid x) \sim \mathrm{N}(Ax, \Omega^{-1})$, then our negative log likelihood can be written in the form given above. Here A is like our feature matrix, Ω is a covariance matrix, and x is like a regression vector.[1] Specify what P, q, and r are for this negative log likelihood.

Solution

The likelihood of the multivariate Gaussian distribution is:

$$L(y|x) = (2\pi)^{n/2}|\Omega|^{1/2} exp\left\{ -\frac{1}{2}(y - Ax)^T \Omega (y - Ax) \right\}$$

Then, the negative likelihood function is:

$$l(y|x) \propto \frac{1}{2}(y - Ax)^T \Omega (y - Ax) = \frac{1}{2}y^T \Omega y + \frac{1}{2}x^t A^T \Omega A x - y^T \Omega A x = \frac{1}{2}x^T P x q^T + r$$

Where we obtained:

$$P = A^T \Omega A$$
$$q = -A^T \Omega y$$
$$r = \frac{1}{2}y^T \Omega y$$

[1] This is notation from the optimization literature, rather than the stats literature. It's important to be able to translate between both.

(C) Let $\phi(x) = \tau\|x\|_1$. Express the proximal operator of this function in terms of the soft-thresholding function that we learned about in the last set of exercises. (An element-wise expression is fine.)

Solution

The proximal operator is:

$$\underset{\gamma}{\text{prox}}\,\phi(x) = \arg\min_z \left\{ \phi(z) + \frac{1}{2\gamma}\|z - x\|_2^2 \right\} = \arg\min_z \left\{ \tau\|x\|_1 + \frac{1}{2\gamma}\|z - x\|_2^2 \right\}$$

$$= \arg\min_z \left\{ \tau \sum_{i=1}^{p} |z_i| + \frac{1}{2\gamma}\|z - x\|_2^2 \right\}$$

Then we have that for each z_i (element-wise):

$$\arg\min_{z_i} \left\{ \tau|z_i| + \frac{1}{2\gamma}(z_i - x_i)^2 \right\} = \arg\min_{z_i} \left\{ \frac{1}{2}(z_i - x_i)^2 + \gamma\tau|z_i| \right\}$$

Which is equal to the soft-thresholding function in Exercises 5:

$$S_\lambda(y) = \arg\min_\theta \ \frac{1}{2}(y - \theta)^2 + \lambda|\theta|$$

Which we probed to be equal to:

$$S_\lambda(y) = \text{sign}(y) \cdot (|y| - \lambda)_+$$

Thus, the proximal operator can be written as:

$$\underset{\gamma}{\text{prox}}\,\phi(x_i) = \text{sign}(x_i) \cdot (|x_i| - \gamma\tau)_+$$

2 The proximal gradient method

Suppose that we have some objective function that can be expressed as $f(x) = l(x) + \phi(x)$, where $l(x)$ is differentiable but $\phi(x)$ is not. The proximal gradient method is designed for precisely this situation.

Recall from above the idea of forming a local linear approximation to a function at some point 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'll apply this idea to the first term in our objective, $l(x)$. Define

$$l(x) \approx \tilde{l}(x; x_0) = l(x_0) + (x - x_0)^{(t)} \nabla l(x_0) + \frac{1}{2\gamma} \|x - x_0\|_2^2$$

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

$$f(x) \approx \tilde{f}(x; x_0) = \tilde{l}(x; x_0) + \phi(x)\,. \tag{3}$$

(A) Consider the surrogate optimization problem in which we minimize the approximation $\tilde{f}(x; x_0)$ in Equation 3, in lieu of our original objective $f(x)$.

$$\hat{x} = \arg\min_x \left\{ \tilde{l}(x; x_0) + \phi(x) \right\}.$$

Show that the solution to this problem is of the form

$$\hat{x} = \operatorname*{prox}_{\gamma} \phi(u)\,, \quad \text{where} \quad u = x_0 - \gamma \nabla l(x_0)\,. \tag{4}$$

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

Solution

We have that,

$$\hat{x} = \arg\min_x \left\{ \tilde{l}(x; x_0) + \phi(x) \right\}$$

Then,

$$\hat{x} = \arg\min_x \left\{ l(x_0) + (x - x_0)^T \nabla l(x_0) + \frac{1}{2\gamma} \|x - x_0\|_2^2 + \phi(x) \right\}$$

$$= \arg\min_x \left\{ x^T \nabla l(x_0) + \frac{1}{2\gamma} \|x - x_0\|_2^2 + \phi(x) \right\} = \arg\min_x \left\{ \frac{1}{2\gamma} ||x - x_0||_2^2 - \frac{1}{2\gamma} 2x^T (\gamma \nabla l(x_0)) + \phi(x) \right\}$$

$$= \arg\min_x \left\{ \frac{1}{2\gamma} \left[||x - x_0||_2^2 - 2x^T (\gamma \nabla l(x_0)) \right] + \phi(x) \right\}$$

$$= \arg\min_x \left\{ \frac{1}{2\gamma} \left[(x - x_0)^T (x - x_0) - 2(x - x_0)^T (\gamma \nabla l(x_0)) + (\gamma \nabla l(x_0))^T (\gamma \nabla l(x_0)) \right] + \phi(x) \right\}$$

$$= \arg\min_x \left\{ \frac{1}{2\gamma} (x - x_0 + \gamma \nabla l(x_0))^T (x - x_0 + \gamma \nabla l(x_0)) + \phi(x) \right\}$$

$$= \arg\min_x \left\{ \frac{1}{2\gamma} \|x - x_0 + \gamma \nabla l(x_0)\|_2^2 + \phi(x) \right\} = \arg\min_x \left\{ \phi(x) + \frac{1}{2\gamma} \|x - (x_0 - \gamma \nabla l(x_0))\|_2^2 \right\}$$

The we have that,

$$= \operatorname*{prox}_{\gamma} \phi(x_0 - \gamma \nabla l(x_0))$$

$$= \operatorname*{prox}_{\gamma} \phi(u)$$

Where, $u = x_0 - \gamma \nabla l(x_0)$

(B) The *proximal gradient* method is an iterative algorithm in which we repeatedly form the approximation in Equation 3 about the current point, and minimize this surrogate function using Equation 4. Written concisely,

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

Now consider the lasso regression problem:

$$\hat{\beta} = \arg\min_{\beta} \left\{ \|y - X\beta\|_2^2 + \lambda \|\beta\|_1 \right\}\,.$$

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

Now implemement the method and apply it to the diabetes data from last week. Make sure you track the convergence of the algorithm, i.e. the objective values over time. Compare your answers to the answers you get from the package software you used last week (i.e. `glmnet` or scikit-learn).[2]

Solution

We use proximal gradient to minimize the objective function

$$\tilde{f}(\beta; \beta_0) = \tilde{l}(\beta; \beta_0) + \phi(\beta)$$

Where,

$$l(\beta) = \frac{1}{2n}\|y - X\beta\|_2^2 = \frac{1}{2n}(y - X\beta)^T(y - X\beta)$$

$$\nabla l(\beta) = -\frac{1}{n} X^T (y - X\beta)$$

$$\phi(\beta) = \lambda \|\beta\|_1$$

We have that,

$$u^{(t)} = \beta^{(t)} - \gamma^{(t)} \nabla l(\beta^{(t)})$$

Then, the update is:

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

We also have that,

$$\beta^{(t+1)} = \underset{\gamma^{(t)}}{\text{prox}}\, \phi(u^{(t)}) \beta^{(t+1)} = \underset{\gamma^{(t)}}{\text{prox}}\, \lambda ||u^{(t)}||_1$$

Where element-wise we have,

$$\beta_j^{(t+1)} = sign(u_j^{(t)}) \cdot (|u_j^{(t)}| - \lambda\gamma^{(t)})_+$$

Which corresponds to the soft-thresholding function $S_\lambda(u_j^{(t)})$.

[2]Keep in mind that those packages are using a value of λ that is rescaled by a factor of n, since they use the loss function

$$\frac{1}{2n}\|y - X\beta\|_2^2 + \lambda\|\beta\|_1\,.$$

Pseudo-code for using the proximal gradient algorithm:

For i in 2 to iteration

- Estimate the gradient $\nabla l(\beta) = -\frac{1}{n} X^T (y - X\beta)$
- Estimate $u^{(t)} = \beta^{(t)} - \gamma^{(t)} \nabla l(\beta^{(t)})$
- Update $\beta_j^{(t+1)} = S_\lambda(u_j^{(t)})$

The main computational cost is present in obtaining the gradient due to the matrix multiplication.

The method implementation is presented in the file $Part\_2.R$,

```r
#-----------------------------------------------------------------------
# Estimate the Lasso Objective Function

Lasso_OF <- function(beta, Y, X, lambda){
  # Compute the Negative Log-Likelihood function
  # Input:
  # beta = regression parameters (vector M)
  # X = features (matrix M x P)
  # Y = vector of observations (vector M)
  # lambda = complexity parameter (scalar)
  # Output:
  # Lasso_OF = Lasso Objective Function (scalar)
  t <- X %*% beta
  OF <- (0.5/nrow(X))*crossprod(Y - t) + lambda * sum(abs(beta))
  OF <- as.numeric(OF)
}

#-----------------------------------------------------------------------
# Estimate the Gradient of the Negative Log-Likelihood

Gradient <- function(beta, Y, X){
  # Estimate the gradient of  the Negative Log-Likelihood of the Logit Model
  # Input:
  # beta = regression parameters (vector M)
  # X = features (matrix M x P)
  # Y = vector of observations (vector M)
  # Output:
  # gradient = gradient (vector M)
  t <- Y - X %*% beta
  grad <- -(1/nrow(X)) * crossprod(X, t)
}

#-----------------------------------------------------------------------
# Estimate the Soft-Thresholding Function
S_lambda <- function(Y, lambda){
  # Estimate beta using the soft-thresholding operator
  # Input:
  # Y = vector of observations (vector M)
  # lambda = complexity parameter (constant)
  # Output:
  # beta = estimated beta vector (vector M)
  t <- cbind(rep(0, length(Y)), abs(Y) - lambda)
```

```r
  prox <- sign(Y) * apply(t, 1, max)
}

#-----------------------------------------------------------------
# The Proximal Gradient Method Implementation

proximal_gradient <- function(Y, X, lambda, gamma, iter, tol){
  # Estimate Lasso regression using The Proximal Gradient Method
  # Input:
  # Y = vector of observations (vector M)
  # X = features (matrix M x P)
  # lambda = complexity parameter (constant)
  # gamma = proximal operator parameter (scalar)
  # iter = number of iterations (scalar = N)
  # tol = tolerance
  # Output:
  # beta = estimated beta vector (vector P)
  # Lasso_OF = negative log-likelihood per iteration (vector N)

  P <- dim(X)[2]
  betas <- array(NA, dim=c(iter, P))
  betas[1,] <- rep(0, P) # Initial guess is zero
  OF <- array(NA, dim = iter)
  OF[1] <- Lasso_OF(betas[1,], Y, X, lambda)

  for (i in 2:iter){
    gradient <- Gradient(betas[i-1,], Y, X) # Step 1 of my pseudocode
    u <- betas[i-1,] - gamma * gradient # Step 2 of my pseudocode
    betas[i,] <- S_lambda(u, gamma * lambda)  # Step 3 of my pseudocode
    OF[i] <- Lasso_OF(betas[i,], Y, X, lambda)

    # Convergence Check
    e <- abs(OF[i-1] - OF[i]) / (OF[i-1] + tol)
    if ( e < tol){
      cat('Converged at ', i)
      OF <- OF[1:i]
      betas <- betas[1:i, ]
      break
    }
    else if ((i == iter) & (e) >= tol){
      print('Did not Converge')
      break
    }
  }
  return(list("Lasso_OF" = OF, "betas" = betas[i,]))
}

#-----------------------------------------------------------------
```

(C) A cool variation on proximal gradient is called the *accelerated* proximal gradient algorithm. The following scheme (due to Nesterov, who's super famous in this area) involves a simple extrapolation step based on the previous iteration:

$$x^{(t+1)} = \underset{\gamma^{(t)}}{\text{prox}}\, \phi(u^{(t)}), \quad u^{(t)} = z^{(t)} - \gamma^{(t)} \nabla l(z^{(t)})$$
$$s^{(t+1)} = \frac{1 + (1 + 4s_t^2)^{1/2}}{2}$$
$$z^{(t+1)} = x^{(t+1)} + \left(\frac{s^{(t)} - 1}{s^{(t+1)}} \right) (x^{(t+1)} - x^{(t)}).$$

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

Note: you can use this acceleration trick in ordinary gradient descent, too.

Solution

Pseudo-code for using the accelerated proximal gradient algorithm:

For i in 2 to iteration

- Estimate the gradient $\nabla l(\beta) = -\frac{1}{n} X^T (y - X\beta)$
- Estimate $s^{(t+1)} = \frac{1+(1+4s_t^2)^{1/2}}{2}$
- Estimate $u^{(t)} = z^{(t)} - \gamma^{(t)} \nabla l(\beta^{(t)})$
- Update $\beta_j^{(t+1)} = S_\lambda(u_j^{(t)})$
- Estimate $z^{(t+1)} = x^{(t+1)} + \left(\frac{s^{(t)}-1}{s^{(t+1)}} \right) (x^{(t+1)} - x^{(t)})$

The method implementation is presented in the file $Part\_2.R$,

```
#-----------------------------------------------------------------------
# The Accelerated Proximal Gradient Method Implementation

ac_proximal_gradient <- function(Y, X, lambda, gamma, iter, tol){
  # Estimate Lasso regression using The Accelerated Proximal Gradient Method
  # Input:
  # Y = vector of observations (vector M)
  # X = features (matrix M x P)
  # lambda = complexity parameter (constant)
  # gamma = proximal operator parameter (scalar)
  # iter = number of iterations (scalar = N)
  # tol = tolerance
  # Output:
  # beta = estimated beta vector (vector P)
  # Lasso_OF = negative log-likelihood per iteration (vector N)

  P <- dim(X)[2]
  betas <- array(NA, dim=c(iter, P))
  betas[1,] <- rep(0, P) # Initial guess is zero
  OF <- array(NA, dim = iter)
  OF[1] <- Lasso_OF(betas[1,], Y, X, lambda)
  s <- rep(NA, iter)
  s[1] <- 1
  z <- rep(0, P)

  for (i in 2:iter){
    gradient <- Gradient(betas[i-1,], Y, X) # Step 1 of my pseudocode
    u <- z - gamma * gradient # Step 2 of my pseudocode
    betas[i,] <- S_lambda(u, gamma * lambda)  # Step 3 of my pseudocode
    s[i] <- (1 + sqrt(1 + 4 * s[i-1]^2))/2 # Step 4 of my pseudocode
    # Step 5 of my pseudocode
    z <- betas[i, ] + ((s[i-1] - 1)/s[i]) * (betas[i, ] - betas[i-1, ])
    OF[i] <- Lasso_OF(betas[i,], Y, X, lambda)

    # Convergence Check
    e <- abs(OF[i-1] - OF[i]) / (OF[i-1] + tol)
    if ( e < tol){
      cat('Converged at ', i)
      OF <- OF[1:i]
      betas <- betas[1:i, ]
      break
    }
    else if ((i == iter) & (e) >= tol){
      print('Did not Converge')
      break
    }
  }
  return(list("Lasso_OF" = OF, "betas" = betas[i,]))
}
```

Convergence Comparison

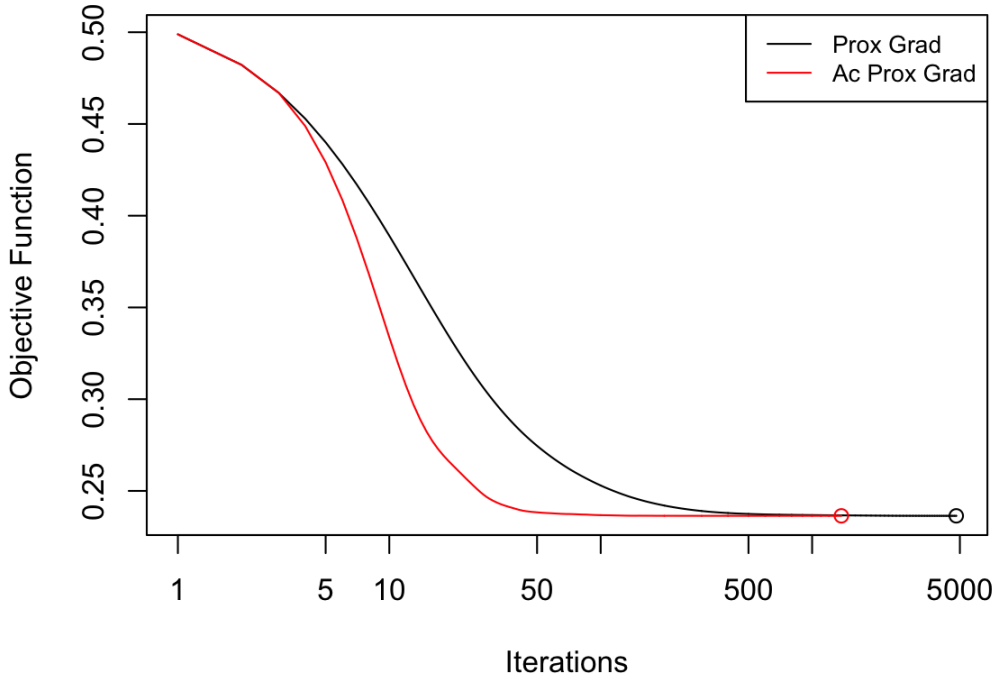

Figure 2: Proximal Gradient Convergency Comparison