Tarea 2

Felipe González Casabianca

Noviembre 10 del 2016

Desarrollo

1. Primero, considere $\phi : \mathbb{R} \to \mathbb{R}$ definida como:

$$\phi(x) = |x|$$

Recuerde que esta función es diferenciable en $\mathbb{R}^*$ así que para cualquier $x_0 \neq 0$, se tiene que:

$$\partial\phi(x_0) = \begin{cases} -1 & \text{si } x < 0 \\ 1 & \text{si } x > 0 \end{cases}$$

y para $x_0 = 0$ note que para cualquier $g \in [-1, 1]$ y $x \in \mathbb{R}$ tenemos que:

$$g(x - x_0) = g\,x \leq |g\,x| = |g||x| \leq |x| = \phi(x) - \phi(x_0)$$

así que:

$$\partial\phi(0) = [-1, 1]$$

y en conclusión:

$$\partial\phi(x_0) = \begin{cases} -1 & \text{si } x < 0 \\ \text{[-1,1]} & \text{si } x = 0 \\ 1 & \text{si } x > 0 \end{cases}$$

Además, como $\phi(x)$ es c.c.p, por el **Lema 3.1.9** de [1] tenemos para $x \in \mathbb{R}^p$ que:

$$\partial\big(\|x\|_1\big) = \partial\left(\sum_{i=1}^{n} \phi(x^T e_i)\right) = \sum_{i=1}^{n} \partial\phi(x_i)\, e_i$$

Ahora bien, sea $t > 0$, $\lambda > 0$ y $h(x) = \lambda\|x\|_1$, y declare:[1]

$$\psi(x, z) = \lambda\|z\|_1 + \frac{1}{2t}\|z - x\|_2^2 \tag{1}$$

así que:

$$prox_{th}(x) = \arg\min_z(\psi(x, z)) \tag{2}$$

Pero recuerde que z es un óptimo de $\psi(x, z)$ (con x fijo) si:

$$\begin{aligned}
0 \in \partial\big(\psi(x,z)\big) &= \partial\left(\lambda\|z\|_1 + \frac{1}{2t}\|z - x\|_2^2\right) \\
&= \partial\big(\lambda\|z\|_1\big) + \partial\left(\frac{1}{2t}\|z - x\|_2^2\right) \quad \text{(ya que ambas son c.c.p)} \\
&= \lambda\sum_{i=1}^{n}\partial\phi(z_i)e_i + \nabla\left(\frac{1}{2t}\|z - x\|_2^2\right) \quad \text{(la norma 2 es diferenciable)} \\
&= \lambda\sum_{i=1}^{n}\partial\phi(z_i)e_i + \frac{1}{t}(z - x)
\end{aligned}$$

Por lo tanto si $\psi(x, z^*)$ es el mínimo, componente a componente, se tiene que para algún valor $p \in \partial\phi(z_i^*)$ que:

$$0 = \lambda\, p + \frac{1}{t}(z_i^* - x_i) \;\Rightarrow\; 0 = p + \frac{1}{\lambda t}(z_i^* - x_i) \Rightarrow\; z_i^* = x_i - \lambda t\, p$$

Lo cual nos lleva a tres casos:

- $x_i > \lambda t$
  Note que si $0 \geq z_i^*$, pasa que:

  $$0 \geq x_i - \lambda t\, p \;\Rightarrow\; \lambda t\, p \geq x_i$$

  Recuerde que $p \in [-1, 1]$, así que en todo caso se tiene que:

  $$\lambda t \geq x_i$$

  Lo cual contradice la hipótesis del caso. Así que $0 < z_i^*$, lo que implica que $p = 1$ y en consecuencia:

  $$z_i^* = x_i - \lambda t$$

[1] A pesar de que el ejercicio exige calcular la función prox para $\hat{h}(x) = \|x\|$, esta es un caso particular de la anterior ($\lambda = 1$), y además nos ahorramos trabajo para el punto 2 en el que será utilizada la función prox de la norma multiplicada por lambda

- $x_i < -\lambda t$
  Procediendo de manera análoga al caso anterior, se tiene que:

$$z_i^* = x_i + \lambda t$$

- $x_i \in [-\lambda t, \lambda t]$

  Note que si $0 < z_i^*$, tenemos que:

$$0 < x_i - \lambda t \ \Rightarrow \ \lambda t \ < x_i$$

  y si $0 > z_i^*$ se obtiene que:

$$0 > x_i + \lambda t \ \Rightarrow \ x_i < -\lambda t$$

  Las cuales son contradicciones, así que necesariamente:

$$z_i^* = 0$$

Por lo tanto, en general se tiene que:

$$prox_{th}(x)_i = \begin{cases} x_i - \lambda t & \text{si } x_i > \lambda t \\ 0 & \text{si } x_i \in [-\lambda t, \lambda t] \\ x_i + \lambda t & \text{si } x_i < -\lambda t \end{cases}$$

2. Recuerde que la función principal es:

$$f(\beta) = \sum_{i=1}^{n} \Big(- y_i x_i^T \beta + log(1 + exp(x_i^T \beta) \Big)$$

Por lo tanto, su gradiente es de la forma

$$\nabla f(\beta) = \sum_{i=1}^{n} \left(- y_i \, x_i + \frac{exp(x_i^T \beta)}{1 + exp(x_i^T \beta)} \, x_i \right)$$

Procedemos a desarrollar cada uno de los algoritmos solicitados:

Nota: el criterio de parada para todos los algoritmos es el mismo. Si $\omega_k = h(x_k) - h(x_{k-1})$, entonces los algorimtos paran cuando el promedio de los últimos 25 ω_k es menor a 0,0001. Adicionalmente a esto, todos los datos fueron normalizados por columnas, para que cada entrada solo tome valores en $[0, 1]$, evitando que se evaluen exponenciales de números muy grandes

Subgradiente

Note que la función f es continua y diferenciable en todo $\mathbb{R}^p$, así que:

$$\partial f(\beta) = \nabla f(\beta)$$

Además, si $\lambda > 0$ y $\phi(x)$ es como en el punto anterior, de este y el **Lema 3.1.9** de [1] tenemos para $x \in \mathbb{R}^p$ que:

$$\partial\big(\lambda\|x\|_1\big) = \partial\left(\sum_{i=1}^{n} \lambda\phi(e_i^T x)\right) = \sum_{i=1}^{n} \lambda\partial\phi(x_i)\, e_i$$

Por lo tanto, como f es convexa[2] y continua en todo $\mathbb{R}^p$, se tiene que es cerrada, así que volvemos a utilizar el **Lema 3.1.9** de [1], para obtener que:

$$\partial h(\beta) = \sum_{i=1}^{n} -y_i\, x_i + \frac{exp(x_i^T\beta)}{1 + exp(x_i^T\beta)}\, x_i \;+\; \lambda\partial\phi(x_i)\, e_i$$

La formula anterior será la utilizada para el procedimiento del subgradiente. En el caso donde la entrada $x_i = 0$ y exista mas de un subgradiente para la norma, se escoge un numero aleatorio entre $[-1, 1]$.

Por último, la sucesión que tiende a cero de los α_k fue escogida de la siguiente manera:

$$\alpha_k = \frac{1}{2k}$$

Subgradiente Estocástico

El procedimiento para este caso es el mismo que el anterior, lo único que cambia es la eleccion del subgradiente. La idea de este procedimiento es escoger únicamente una fracción de los x_i de la muestra para computar el gradiente, y así de cierta manera, se consigue una dirección sensata a cambio de menos cálculos computacionales. En particular, en cada iteración de este algoritmo se escogieron **60 %** de los x_i de la muestra, seleccionados **uniformemente**.

Proximal

Ahora, del desarrollo del primer punto tenemos que si $g(x) = \lambda\|x\|_1$, con $\lambda > 0$ tenemos que:

[2] Espero que sea convexa!

$$prox_{tg}(x)_i = \begin{cases} x_i - \lambda t & \text{si } x_i > \lambda t \\ 0 & \text{si } x_i \in [-\lambda t, \lambda t] \\ x_i + \lambda t & \text{si } x_i < -\lambda t \end{cases}$$

Tanto el algoritmo **Proximal**, como el **Proximal Acelerado**, utilizan esta caracterización de la función *prox*.

Proximal Acelerado

La implementación de este algoritmo se hizo sin *Backtracking* ya que resultó siendo mucho mas eficiente con un $\alpha = 0{,}001$ constante.

Resultados

A continuación se muestra la gráfica de el número de iteraciones versus $Log(h(\beta) - h*)$ con $\lambda = 1$

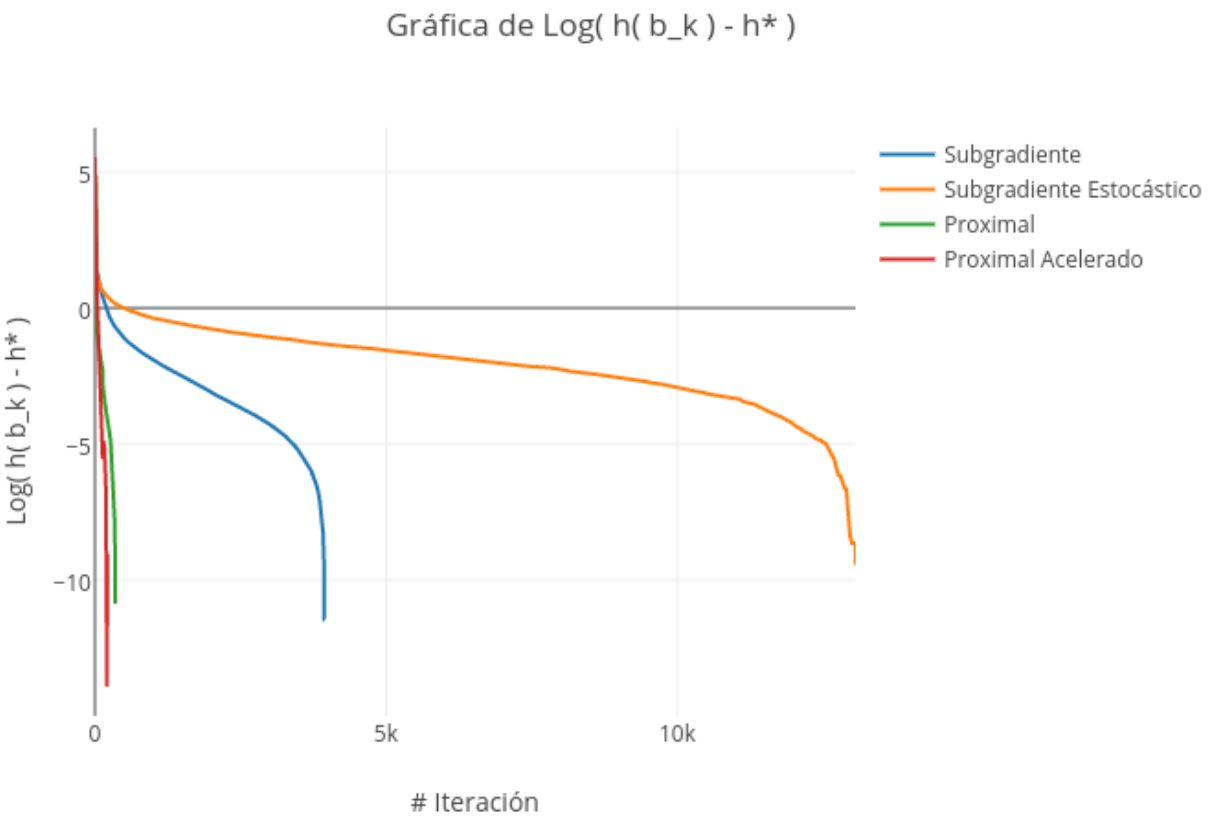

Figura 1: Gráfica comparativa de $Log(h(\beta_k) - h*)$ para cada iteración

Note que todos los métodos tienen un comportamiento similar y que el más eficiente, claramente es el metodo **Proximal Acelerado**. Seguimos con algunas gráficas exploratorias del comportamiento de los algoritmos a medida que crece λ:

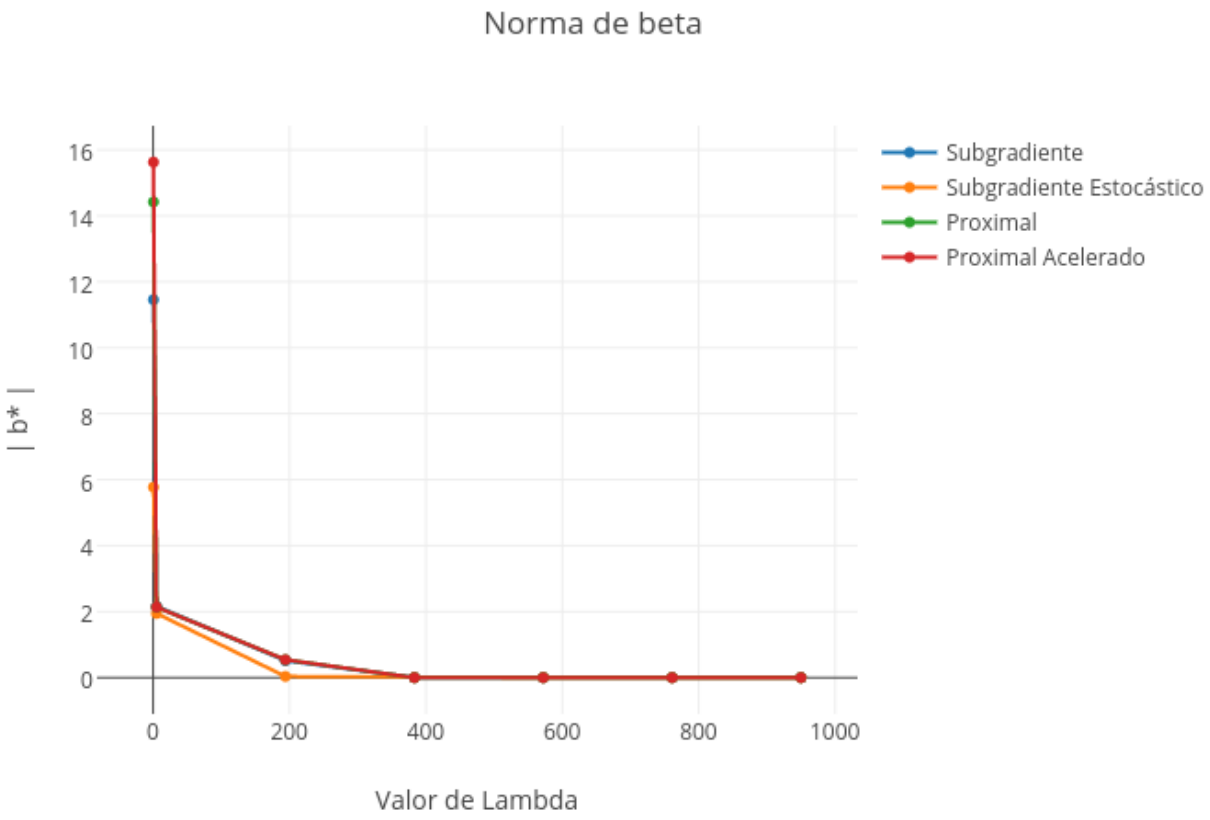

Figura 2: Gráfica de la norma de β^* para cada Lambda

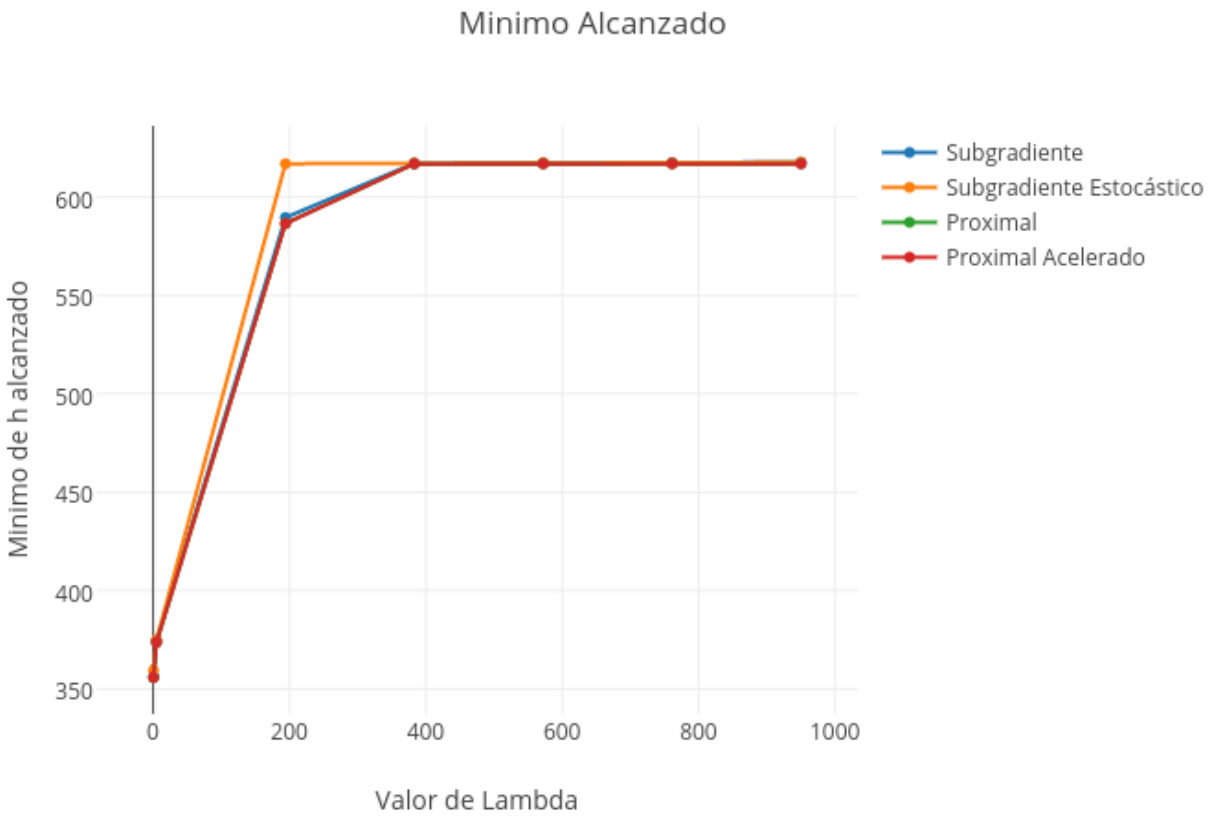

Figura 3: Gráfica de mínimimo $h(\beta)$ alcanzado para cada Lambda

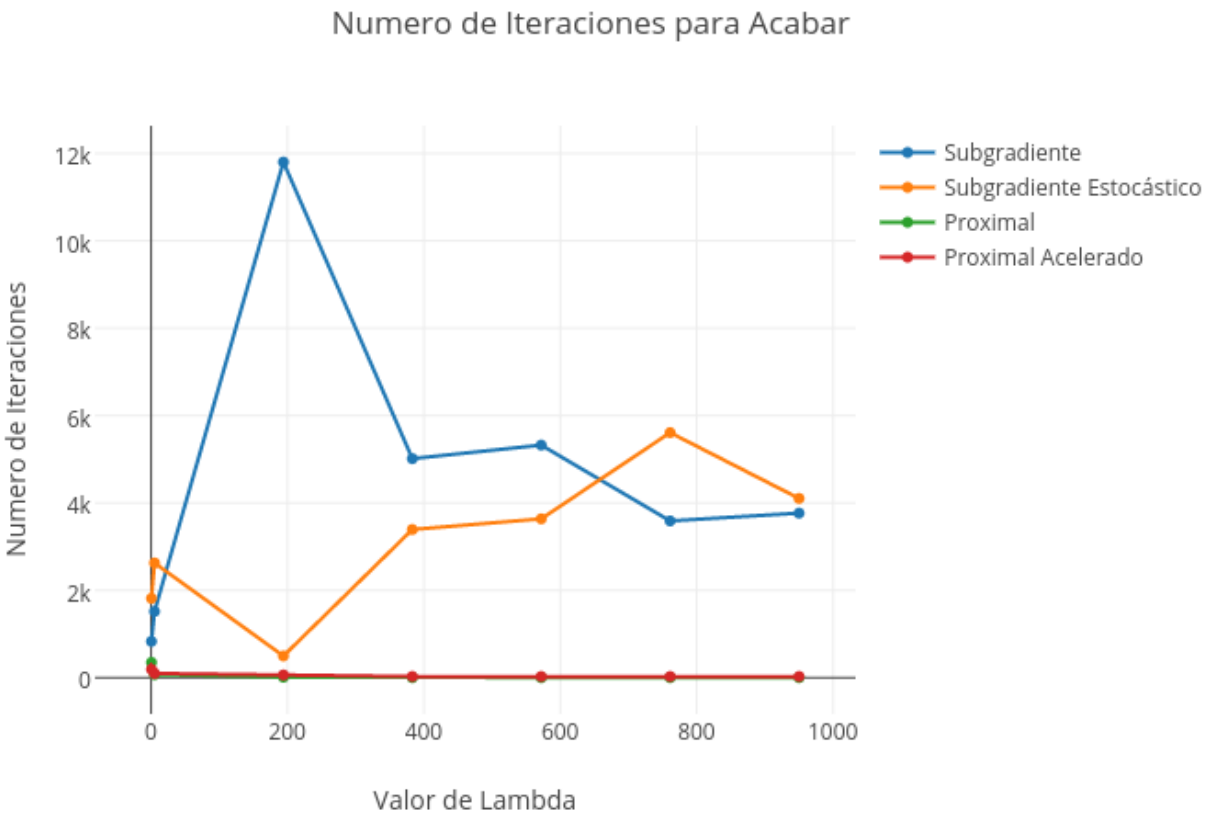

Figura 4: Gráfica del número de iteraciones necesarias para acabar, por cada Lambda

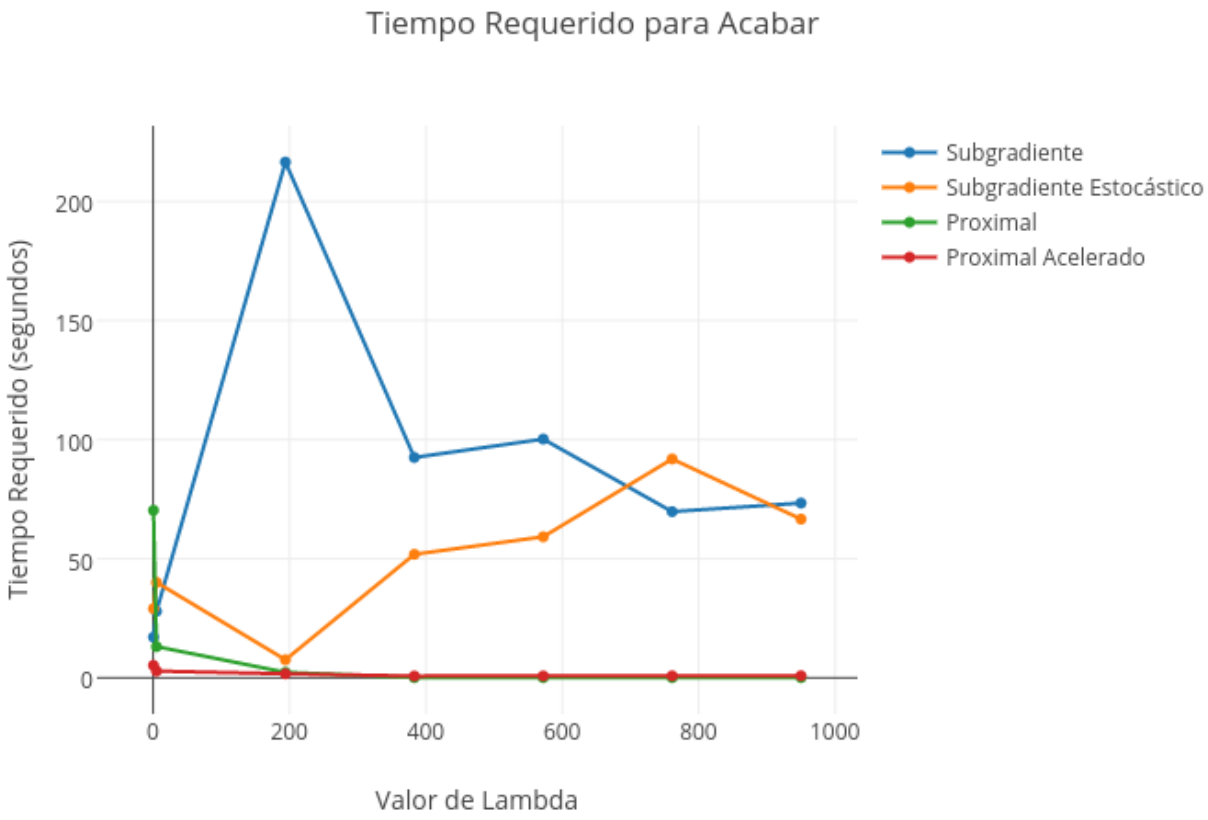

Figura 5: Gráfica del tiempo necesario para acabar, por cada Lambda

De las gráficas anteriores, concluimos sin lugar a duda que el mejor método (independiente de λ) es el **Proximal Acelerado**. Note que no solo es el mas eficiente tanto en tiempo como iteraciones, sino que además reporta los mejores mínimos para cada uno de los λ. También cabe aclarar que todos los métodos reportan valores similares para $\|\beta^*\|_1$, por lo que la decisión de cual es el mejor método para este problema se traduce a un tema de factibilidad computacional.

3. Primero note que si $g(\beta) = \lambda\|\beta\|_1$ y $\hat{t} = 1/n\alpha_k$ se tiene que

$$\beta^{k+1} = prox_{g\,\hat{t}}\left(\bar{\eta}^{k+1} + \bar{\mu}^k\right)$$

Por lo tanto, del punto 1, tenemos que:

$$\beta_i^{k+1} = prox_{g\,\hat{t}}\left(\bar{\eta}^{k+1}+\bar{\mu}^k\right)_i = \begin{cases} \left(\bar{\eta}_i^{k+1}+\bar{\mu}_i^k\right) - \lambda t & \text{si } \left(\bar{\eta}_i^{k+1}+\bar{\mu}_i^k\right) > \lambda t \\ 0 & \text{si } \left(\bar{\eta}_i^{k+1}+\bar{\mu}_i^k\right) \in [-\lambda t, \lambda t] \\ \left(\bar{\eta}_i^{k+1}+\bar{\mu}_i^k\right) + \lambda t & \text{si } \left(\bar{\eta}_i^{k+1}+\bar{\mu}_i^k\right) < -\lambda t \end{cases}$$

Con esto en mente, la actualización de los β se hace a través de la función anterior. Para el caso de los η, se uso el descenso del gradiente con *backtracking* de la tarea pasada para avanzar en cada iteración. Dicho procedimiento fue programado para calcularse en paralelo, utilizando la librería *multiprocessing* de Python.

Sin embargo, es necesario instalar una serie de librerías para que efectivamente el programa haga uso de los distintos núcleos del computador, las cuales no fui capas de hacerlo sobre el servicio de clustering de Amazon Web Service, por lo tanto, la rutina en paralelo solo contaba con 4 núcleos disponibles para hacer el computo (mi computador personal). Por lo tanto, teniendo en cuenta que era necesario realizar una optimización de descenso del gradiente 215 veces por cada iteración (hay 860 filas y cada uno de los cuatro núcleos se puede encargar de una en simultaneo), sigue siendo un proceso muy lento. Cada una de las optimizaciones tomaba alrededor de 4 segundos, lo que traduce a 14 minutos por iteración.

Adicionalmente a esto, suponiendo que el método está bien programado, es un algoritmo muy lento en su convergencia. A continuación tenemos un ejemplo de la gráfica $log(h(\eta_k) - h*)$ con $m = 5$:

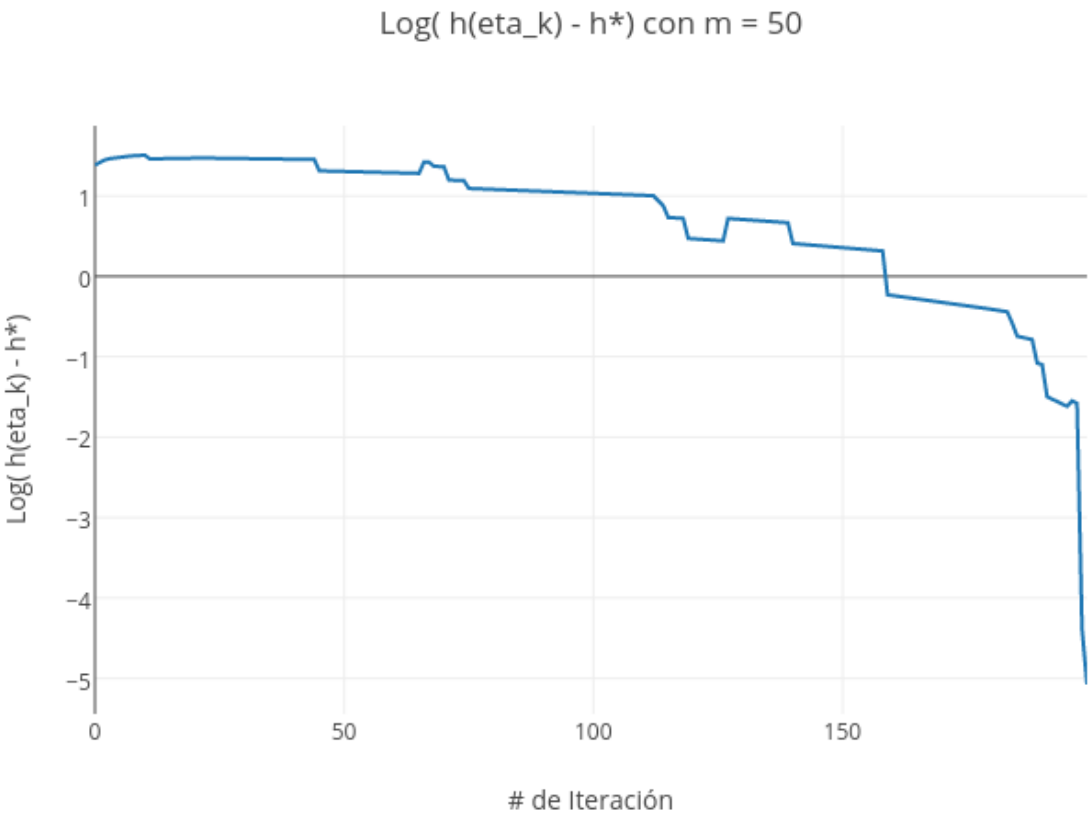

Figura 6: Gráfica de $log(h(\eta_k) - h*)$ con $m = 5$, unicamente las primeras 200 iteraciones

En promedio y para este caso en específico, el mínimo estaba descendiendo en 0.3 unidades por iteración y después de 200 iteraciones se encontraba en 222,72. Teniendo en cuenta que el mínimo para este problema en particular es 27,87 (usando el método proximal acelerado), todavia faltaban muchas iteraciones para acabar.

En resumen, al no poder paralelizar completamente este método (una rutina donde exista un núcleo de computo por cada f_i) no es un algoritmo muy eficiente para solucionar este tipo de problemas.

Código

El codigo fue desarrollado en python 2.7 y se encuentra disponible al público en el link:

`https://github.com/minigonche/subgradient_descent`

Por su extensión, no se incluye en esta entrega

Referencias

[1] Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.