Alternate Directions Method of Multipliers (ADMM)

Ignacio Ramírez

22 de octubre de 2019

1 Introducción

Nota Tomado en parte de la monografía "Proximal Algorithms" de Parikh y Boyd.

Consideremos el problema

$$\min f(x) + g(x) \tag{1}$$

donde $f, g : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ son funciones convexas, cerradas y propias. Cualquiera de ellas, o ambas, pueden ser no suaves. El método *alternating direction method of multipliers* (ADMM) consiste en la siguiente iteración:

$$x^{k+1} = \text{prox}_{\lambda f}(z^k - u^k) \tag{2}$$
$$z^{k+1} = \text{prox}_{\lambda g}(x^{k+1} + u^k) \tag{3}$$
$$u^k = u^k + x^{k+1} - z^{k+1} \tag{4}$$

donde el nombre *alternating directions* sale de que en el primer paso se va hacia $-u^k$ y en el segundo se va hacia u^k. Este método converge bajo condiciones muy amplias.

Discusión Es importante notar que si bien x^k y z^k convergen hacia el mismo punto óptimo, tienen propiedades distintas. Por ejemplo, $x^k \in \text{dom } f$ mientras que $z^k \in \text{dom } g$. Si, por ejemplo, utilizamos g para expresar restricciones implícitas, entonces z^k siempre será factible, mientras que x^k sólo lo será en el límite. Si $g = \|\cdot\|_1$ entonces z^k será esparso porque el operador proximal de la norma ℓ_1 es el soft-thresholding, mientras que x^k solo será *aproximadamente* esparso.

El ADMM ofrece una ventaja clave cuando los operadores proximales de f y g son fáciles de evaluar, pero el de $f + g$ no lo es, algo que sucede muy seguido. El ADMM es particularmente útil como herramienta para paralelizar problemas, como veremos más adelante.

2 ADMM y el Lagrangeano Aumentado

Recordemos la definición de Lagrangeano Aumentado para el problema genérico $\min f(x)$ s.a. $h(x) = 0$,

$$L_c(x, u) = f(x) + u^T h(x) + \frac{c}{2}\|h(x)\|^2. \tag{5}$$

Habíamos visto que uno de los métodos posibles para resolver el problema original en términos del Lagrangeano Aumentado es el *método de los multiplicadores*, que consiste en la siguiente iteración,

$$x^{k+1} = \arg\min L_{c^k}(x, y^k) \tag{6}$$
$$y^{k+1} = y^k + ch(x^{k+1}). \tag{7}$$

El ADMM puede verse como una variante del algoritmo anterior para el caso especial en que $h(x, z) = x - z$. Los primeros dos pasos equivalen a subdividir el primer paso del método de los multiplicadores

en dos, minimizando primero en x y luego en z (esto es un caso particular de la estrategia *block coordinate descent*, que funciona siempre que el objetivo sea separable en los bloques de variables),

$$x^{k+1} = \arg\min_\xi L_{c^k}(\xi, z^k, y^k) \tag{8}$$
$$z^{k+1} = \arg\min_\zeta L_{c^k}(x^{k+1}, \zeta, y^k) \tag{9}$$

Resta ahora ver que los pasos (8)–(9) equivalen a (2)–(3). Sustituimos ahora los subproblemas de cada paso,

$$x^{k+1} = \arg\min_\xi f(x) + (y^k)^T\xi + (c/2)\|\xi - z^k\|_2^2 \tag{10}$$
$$z^{k+1} = \arg\min_\zeta g(z) - (y^k)^T\zeta + (c/2)\|x^{k+1} - \zeta\|_2^2, \tag{11}$$

y completamos cuadrados (ejercio en clase):

$$x^{k+1} = \arg\min_\xi f(x) + (c/2)\|\xi - z^k + (1/c)y^k\|_2^2 \tag{12}$$
$$z^{k+1} = \arg\min_\zeta g(z) + (c/2)\|x^{k+1} - \zeta - (1/c)y^k\|_2^2,, \tag{13}$$

de donde se reconoce la definición de los operadores proximales de parámetro c evaluados en los puntos $z^k - (1/c)y^k$ y $x^k + (1/c)y^k$. Definiendo $u^k = (1/c)y^k$ se ve que estos son precisamente los pasos (2) y (3). Luego dividiendo (7) por c se llega a (4).

2.1 ADMM linearizado

Un caso muy común (en estadística, machine learning o procesamiento de señaneles por ejemplo) se da cuando lo que se busca es

$$\min f(x) + g(Ax)$$

con f y g cerradas, propias y convexas como antes, y $A \in \mathbb{R}^{m\times n}$. Si bien el problema puede resolverse en principio con ADMM definiendo $\hat{g}(x) = g(Ax)$, muy comunmente el operador proximal de $\hat{g}(x)$ se complica muchísimo (no siempre! hay que verificarlo). En este escenario, inspirándose en el nexo de ADMM con el método de los multiplicadores, se puede atacar el problema mediante el método ADMM linearizado, que se define así,

$$\begin{aligned} x^{k+1} &= \text{prox}_{\mu f}(x^k - (\mu/\lambda)A^T(Ax^k - z^k + u^k)) \\ z^{k+1} &= \text{prox}_{\lambda g}(Ax^{k+1} + u^k) \\ u^k &= u^k + Ax^{k+1} - z^{k+1}. \end{aligned}$$

Notar que los últimos dos pasos son análogos al ADMM común utilzando como restricción $Ax = z$ en lugar de $x = z$. El primer paso surge de linearizar el lagrangeano aumentado en x, agregar una penalización cuadrática adicional $(\mu/2)\|x^k - x\|_2^2$ (esto se hace típicamente para mantener x dentro de un rango donde la linearización es una aproximación razonablemente buena) y completar cuadrados. (Se deja como ejercicio de obligatorio derivar este método).

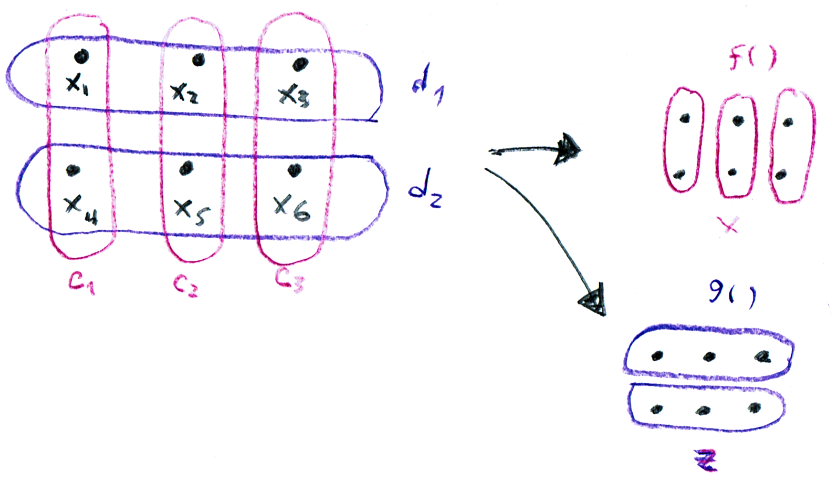

Figura 1: Problema donde la función f es separable en tres subvectores de dimensión 2, y la función g es separable en dos subvectores de dimensión 3. El splitting permite aprovechar la separabilidad a la hora de resolver el problema.

3 Paralelización y computación distribuída con ADMM

3.1 Estructura del problema y notación

Vamos a considerar problemas donde la variable a optimizar es $x \in \mathbb{R}^n$. Llamamos $[n] = \{1, 2, \ldots, n\}$. Dado $c \subseteq [n]$ denotamos como $x_c \in \mathbb{R}^{|c|}$ al subvector de componentes de x indexados por el conjunto c. La colección $\mathcal{P} = \{c_1, c_2, \ldots, c_N\}$ es una partición de $[n]$ sii $\cup_i c_i = [n]$ y $c_i \cap c_j =$ para $i \neq j$. Decimos que una función es $\mathcal{P}$-separable si

$$f(x) = \sum_{i=1}^{N} f_i(x_{c_i}).$$

Decimos que c_i es el *alcance* de f_i. La clave de la separabilidad es que nos permite evaluar el operador proximal sobre cada componente c_i de manera independiente,

$$\big(\text{prox}_f(x)\big)_i = \text{prox}_{f_i}(x_{c_i}).$$

Consideremos ahora el problema

$$\min_x f(x) + g(x), \tag{14}$$

con $g(x)$ propia y convexa y separable en otra partición de $[n]$, $\mathcal{Q} = \{d_1, d_2, \ldots, d_M\}$. El problema se puede escribir entonces como

$$\min_x \sum_{i=1}^{N} f_i(x_{c_i}) + \sum_{j=1}^{M} g_j(x_{c_j}), \tag{15}$$

Para resolver el problema anterior con ADMM hacemos primero el *splitting*, agregando como siempre la variable auxiliar z y la restricción $x = z$,

$$\min_x \sum_{i=1}^{N} f_i(x_{c_i}) + \sum_{j=1}^{M} g_j(z_{c_j}) \text{ sujeto a } \quad x = z \tag{16}$$

y luego planteamos la iteración ADMM,

$$\begin{aligned} x_{c_i}^{k+1} &= \quad \text{prox}_{\lambda f_i}(z_{c_i}^k - u_{c_i}^k), \quad i = 1, \ldots, N \\ z_{d_j}^{k+1} &= \text{prox}_{\lambda g_j}(x_{d_j}^{k+1} - u_{d_j}^k), \quad j = 1, \ldots, M \\ u^{k+1} &= \quad u^k + x^{k+1} - z^{k+1}. \end{aligned}$$

El primer paso puede ejecutarse en paralelo en N pasos independientes, el segundo en M pasos independientes, y el último es trivialmente paralelizable. La figura **??** muestra un ejemplo en donde $N = 3$ y $M = 2$, o dicho de otra manera "x se separa en 3 y z en 2".

Ahora veremos casos particulares interesantes del planteo general recién visto.

3.2 Consenso

Consenso global El problema a resolver tiene la forma

$$\min f(x) = \sum_{i=1}^{N} f_i(x). \tag{17}$$

La idea es minimizar cada una de las funciones "locales" $f_i(x)$ (local en el sentido de un problema distribuido en donde cada nodo puede ver solo f_i) en x de manera independiente. Para esto transformamos al problema en su *forma de consenso*,

$$\begin{array}{ll} \min\limits_{x\{1\},x\{2\},\ldots,x\{N\}} & \sum_{i=1}^{N} f_i(x_i) \\ \text{sujeto a} & x\{1\} = x\{2\} = \ldots = x\{N\}. \end{array}$$

La idea es crear N "copias" de x y luego hacerlas coincidir mediante la restricción de consenso o *consistencia*. Como siempre en ADMM, la restricción será incorporada al problema como una función $g()$, indicatriz del *conjunto de consenso*

$$\mathcal{C} = \{\ (x\{1\}, x\{2\}, \ldots, x\{N\}) : x\{1\} = x\{2\} = \ldots = x\{N\}\ \}$$

El problema queda entonces

$$\min_{x\{1\},x\{2\},\ldots,x\{N\}} \quad \sum_{i=1}^{N} f_i(x\{i\}) + I_{\mathcal{C}}(x\{1\}, x\{2\}, \ldots, x\{N\}).$$

En términos del planteo general de la figura 1, el problema es ahora sobre

$$y = (x\{1\}, x\{2\}, \ldots, x\{N\}) \in \mathbb{R}^{nN}.$$

La figura 2 muestra las particiones $\mathcal{P}$ y $\mathcal{Q}$ de $f(y) = \sum_i f_i(x_i)$ y $g(y) = I_{\mathcal{C}}(y)$ para este problema, donde cada columna de N puntos es una copia de x.

Siendo que $g(\cdot)$ es la función indicatriz del conjunto $\mathcal{C}$, el operador proximal correspondiente es la proyección sobre $\mathcal{C}$, que no es otra cosa que (se deja como ejercicio al lector),

$$z\{i\}^{k+1} = \bar{z},\ i = 1, \ldots, N, \quad \bar{z}^k = \frac{1}{N} \sum_i z\{i\}^k.$$

Podemos entonces escribir el ADMM de manera más compacta para este caso,

$$\begin{array}{rl} x_i^{k+1} = & \text{prox}_{\lambda f_i}(\bar{x}_i^k - u_i^k), \quad i = 1, \ldots, N \\ u_i^{k+1} = & u_i^k + x^{k+1} - \bar{x}^{k+1}. \end{array}$$

Consenso general En este caso (continuando con la jerga de algoritmos distribuidos), cada "nodo local" f_i tiene acceso sólo a un subconjunto c_i de las variables de x, y sólo se pretende que las copias i e i' coincidan en $c_i \cap c_{i'}$. (El caso global visto anteriormente corresponde a $c_i = [n]$ para todo $i = 1, \ldots, N$.)

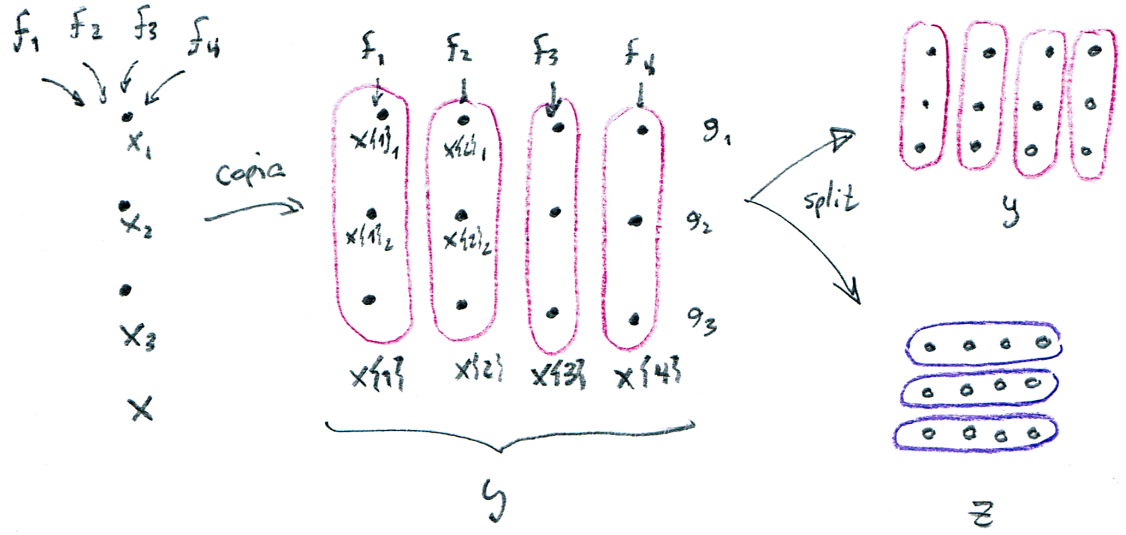

Figura 2: Estructura del problema de consenso global. En este caso se generan 4 copias de la variable $x \in \mathbb{R}^3$; esta pasa a ser una variable $y \in \mathbb{R}^{12}$. Luego se realiza el splitting $y = z$ y se impone el consenso en z, que en este caso tiene una solución muy simple.

3.3 Intercambio o mercadeo

El problema de mercadeo (*exchange*) se origina en problemas económicos de modelado de mercados donde hay N operadores del mercado, cada uno manejando un conjunto de bienes x_i ya sea comprando el bien j ($(x_i)_j$ positivo) o vendiendo (negativo), y debe existir un balance de los bienes comprados y vendidos,

$$\begin{array}{rl} \min & \sum_i f_i(x_i) \\ \text{sujeto a} & \sum_i x_i = 0 \end{array}$$

El problema se plantea de manera idéntica al anterior, con $g = I_{\mathcal{C}}$, siendo $\mathcal{C}$ el conjunto de *clearing* o *equilibrio*,

$$\mathcal{C} = \{(x\{1\}, x\{2\}, \ldots, x\{N\}) \in \mathbb{R}^{nN} : x\{1\} + x\{2\} + \ldots + x\{N\} = 0\}.$$

Resta ver cómo es la proyección sobre el conjunto $\mathcal{C}$,$\Pi_{\mathcal{C}}$, en este caso. Resulta (se deja como ejercicio) que también es muy sencilla, y equivale a "eliminar la DC" (en términos de ingeniería eléctrica),

$$(\Pi_{\mathcal{C}}(v\{1\}, v\{2\}, \ldots, v_N))_i = v_i - \bar{v}, \quad \bar{v} = (1/N)\sum_i v\{i\}. \tag{18}$$

De nuevo esto permite compactar el ADMM,

$$\begin{array}{rl} x_i^{k+1} = & \text{prox}_{\lambda f_i}(x_i^k - \bar{x}_i^k - u_i^k), \quad i = 1, \ldots, N \\ u_i^{k+1} = & u_i^k + \bar{x}^{k+1}. \end{array}$$

Mercadeo generalizado En este caso también podemos considerar un equilibrio parcial, asumiendo que cada "agente" participa sólo de un subconjunto de los bienes, y que el balance de cada bien se da solo entre aquellos agentes que participan en él. La solución es muy similar a (18). Se deja como ejercicio.

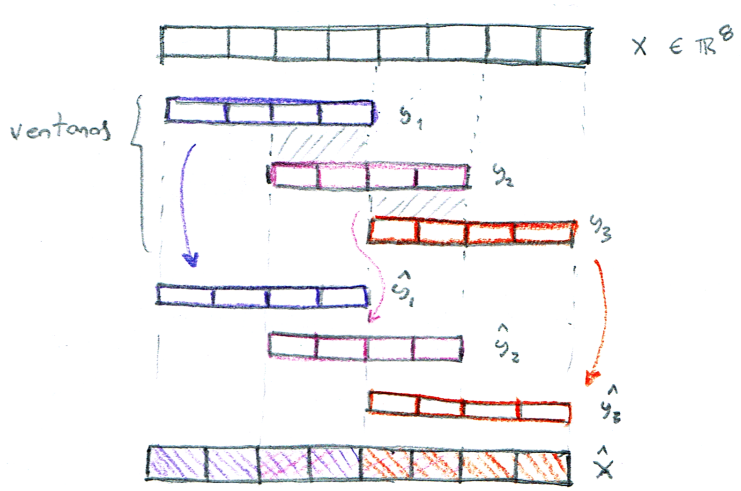

Figura 3: Esquema tradicional de procesamiento por ventanas. De arriba a abajo: señal original ($n = 8$; descomposición en tres ventajas de 4 muestras c/u, con un solapamiento del 50% (2 muestras); pegado de ventanas en su lugar original. Las muestras 4,5,6,7 tienen dos valores posibles cada una y debe decidirse su valor final mediante algún criterio.

4 Ejemplo: algoritmo PACO

El siguiente ejemplo muestra un uso práctico del concepto de consenso parcial. El problema es el siguiente: tenemos una *señal*, en este ejemplo un vector unidimensional, que queremos procesar de alguna manera. Muchos métodos de procesamiento de señales (por ejemplo, procesamiento de audio) procesan las señales de a *ventanas* (o parches–patches en inglés): en lugar de procesar la señal entera, se corta a ésta en fragmentos contiguos (ventanas) generalmente solapados, se procesa cada fragmento por separado, y luego se vuelven a *pegar* los fragmentos en su lugar correspondiente. La figura **??** muestra un esquema de lo anterior.

Cuando las ventanas se solapan (como en el ejemplo), habrá muestras que serán representadas en más de una ventana. El problema que surge aquí es que, luego del procesamiento, las ventanas procesadas pueden tener valores finales distintos de esas muestras que aparecen múltiples veces. A la hora de pegar las ventanas procesadas para armar el resultado final, debe decidirse entonces qué valor poner en aquellos lugares que tienen más de un candidato como resultado.

La forma tradicional de hacer lo anterior es simplemente promediar, como lo muestra el ejemplo de la figura 3. Esto, sin embargo, no siempre es lo ideal.

El algoritmo PACO (PAtch COnsensus) surge de la observación anterior: la idea es eliminar las discrepancias antes de pegar las ventanas, de modo que todas las ventanas tengan la misma estimación de una misma muestra en la señal original. En otras palabras, se busca que las ventanas tengan consenso entre ellas en los lugares que se solapan. Este problema, ilustrado en la figura 4, es un caso del consenso parcial que discutimos anteriormente.

Un problema en donde PACO ha resultado efectivo es en el de completar datos faltantes en señales (algo que en procesamiento de imágenes se conoce también como *inpainting*). En este caso, la función que busca minimizarse es la norma ℓ_1 de los coeficientes de la transformada DCT de cada una de las ventanas (Discrete Cosine Transform, una variante de la transformada de Fourier para señales reales). Si tenemos una señal descompuesta en N ventanas de largo M cada una, la función $f()$ se define de la siguiente manera:

$$f(y_1, y_2, \ldots, y_N) = \sum_{j=1}^{N} f_j(y_j)\,, \quad f_j(y_j) = \sum_{i=1}^{M} w_i |(Dy_j)_i|,$$

donde w_i son M parámetros inversamente proporcionales a la magnitud absoluta promedio de la

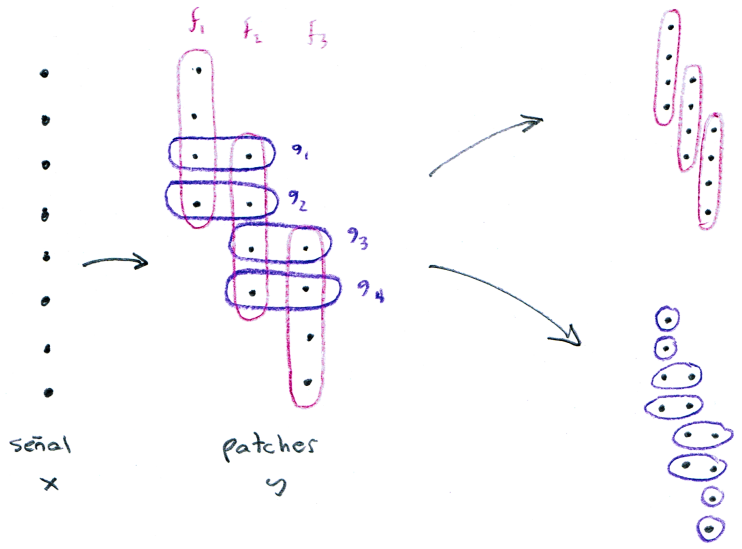

Figura 4: Algoritmo PACO y su estructura según el esquema visto hasta ahora. Notar que es un problema de consenso parcial, ya que el consenso que se exige para cada coordenada involucra sólo a algunas de las muestras.

coordenada i de los vectores $a_j = Dy_j$, y D es la representación matricial de la transformada DCT.

La figura 5 muestra un ejemplo del método PACO-DCT tomado del pre-print del paper `https://arxiv.org/abs/1808.06942`, en donde además pueden encontrarse más detalles sobre el método en cuestión y su aplicación a otros tipos de señales.

Figura 5: Ejemplo de resultado de PACO en inpainting de imágenes. Las manchas blancas son regiones en donde los datos son desconocidos (las regiones en sí son conocidas). El resultado del algoritmo se muestra a la izquierda.