Ejercicio 1

Parte a

Sea

$$I_C(x) = \begin{cases} 0 & , \quad x \in C \\ +\infty & , \quad x \notin C \end{cases}$$

Queremos probar que $\text{prox}_{I_C}(x) = \Pi_C(x)$, con

$$\Pi_C(x) = \min_{z \in C} \|x - z\|$$

Recordando que el proximal se define como

$$\text{prox}_{\lambda f}(v) = \arg\min \left(f(x) + \frac{1}{2\lambda} \|x - v\|_2^2 \right)$$

En nuestro caso particular se tiene que:

$$\text{prox}_{I_C}(v) = \arg\min \left(I_C(x) + \|x - v\|_2^2 \right) \tag{1}$$

Observar que el x que minimiza la ecuación 1 debe pertenecer a C ($x \in C$), ya que si esto **no** ocurriera $I_C(x) = +\infty$ generando que el valor de la función a optimizar sea $+\infty$ ya que $\|x - v\|_2^2 < \infty$.
Es decir, el problema anterior es equivalente a introducir la restricción anterior

$$\text{prox}_{I_C}(v) = \arg\min_{x \in C} \|x - v\|_2^2$$

$$\Rightarrow \text{prox}_{I_C}(v) = \arg\min_{x \in C} \|v - x\|_2^2 = \Pi_C(v)$$

Con la ecuación anterior queda probado lo solicitado.

Parte b

El método de integración híbrido forward–backward y la aproximación dada esta dada por las siguientes ecuaciones

$$\begin{cases} \frac{x^{k+1} - x^k}{h} = -\nabla f \left(\frac{x^{k+1} + x^k}{2} \right) \\ \nabla f \left(\frac{x^{k+1} + x^k}{2} \right) \approx \frac{\nabla f\left(x^{k+1}\right) + \nabla f\left(x^k\right)}{2} \end{cases}$$

Combinando ambas ecuaciones se tiene

$$\frac{x^{k+1} - x^k}{h} \approx -\frac{\nabla f\left(x^{k+1}\right) + \nabla f\left(x^k\right)}{2}$$

Operamos en la ecuación anterior

$$\Rightarrow x^{k+1} - x^k \approx -\frac{h}{2} \nabla f\left(x^{k+1}\right) - \frac{h}{2} \nabla f\left(x^k\right)$$

Continuamos operando

$$\Rightarrow x^{k+1} + \frac{h}{2}\nabla f\left(x^{k+1}\right) \approx x^k - \frac{h}{2}\nabla f\left(x^k\right) \tag{2}$$

Recordar que la condición de optimalidad, dada en el teórico, indica que

$$x = \text{prox}_{\lambda f}(v) \Leftrightarrow v \in x + \lambda \partial f(x)$$

En el caso de que f sea diferenciable el pertenece ($\in$) se convierte en igualdad ($=$). Por lo que la condición es

$$x = \text{prox}_{\lambda f}(v) \Leftrightarrow v = x + \lambda \nabla f(x) \tag{3}$$

Llamemos en la Ecuación 2 a x, v y λ como sigue

$$\begin{cases} x = x^{k+1} \\ v = x^k - \lambda \nabla f\left(x^k\right) \\ \lambda = \frac{h}{2} \end{cases} \tag{4}$$

Con las variables como en 4, notar que tenemos el termino derecho de la condición de optimalidad ($v = x + \lambda \nabla f(x)$). De esta forma usando 3 se llega a

$$\boxed{x^{k+1} = \text{prox}_{\frac{h}{2}f}\left(x^k - \frac{h}{2}\nabla f\left(x^k\right)\right)}$$

La expresión enmarcada representa el método iterativo de optimización. Notar que este método es exactamente el *Proximal Gradient Descent (PGD)*.

Ejercicio 2

Parte a

Se implemento *Proximal Gradient Descent*, para esto se comenzó desde el punto $x^0 = (0, 0)$ y se escogió como condición de parada que $|f(x^{k+1}) - f(x^k)| < 0.0001$. De esta forma, se obtuvo como punto óptimo

$$x^* = \begin{bmatrix} -0.1327 \\ 0.1270 \end{bmatrix}$$

El total de iteraciones insumidas para que se cumpla la condición de parada fue 35. Para tomar la medida del tiempo, se decidió promediar durante 10.000 corridas. Mientras que el tiempo insumido fue 2.982 ms

Parte b

1.

En el Ejercicio 1 vimos que si una función f es diferenciable y convexa se cumple lo siguiente

$$x = \text{prox}_{hf}(v) \Leftrightarrow v = x + h\nabla f(x)$$

Se tiene que $\nabla f(x) = A^T(Ax - b)$. Por lo tanto el proximal de v debera cumplir que

$$v = x + hA^T(Ax - b)$$

Se opera sobre la ecuación anterior con el objetivo de despejar $x(v)$ y así obtener el $\text{prox}_{hf}(v)$.

$$\Rightarrow v = x + hA^TAx - hA^Tb$$

$$\Rightarrow v + hA^Tb = (Id + hA^TA)x$$

$$\Rightarrow x = \left(Id + hA^TA\right)^{-1}\left(v + hA^Tb\right)$$

Por lo que el operador proximal sera

$$\boxed{\text{prox}_{hf}(v) = \left(Id + hA^TA\right)^{-1}\left(v + hA^Tb\right)} \tag{5}$$

2.

Para obtener la solución usando *ADMM*, se escogieron los mismos criterios que con *PGD*. Es decir, $x^0 = (0, 0)$ y como condición de parada que $|f(x^{k+1}) - f(x^k)| < 0.0001$. En cuanto al α, se escogieron 2 valores: $\alpha_1 = 1/\left\|A^TA\right\|_2$ y $\alpha_2 = 10/\left\|A^TA\right\|_2$. Los puntos óptimos obtenidos para cada α fueron

$$x_1^* = \begin{bmatrix} -0.1327 \\ 0.1269 \end{bmatrix}, x_2^* = \begin{bmatrix} -0.1327 \\ 0.1272 \end{bmatrix}$$

El total de iteraciones requeridas fue 40 y 9 respectivamente. Mientras que los tiempos fueron $3.445ms$ y $0.715ms$.
De estos resultados se pueden apreciar dos cosas, los sensible que es ADMM al paso ya que en un caso tarda x4 más iteraciones que en el otro. Sin embargo, también parece ser robusto ya que igualmente converge al punto óptimo esperado, se probo con otros valores de α llegando en todos los casos a puntos cercanos al óptimo (se puede ver en el código).

Parte c

Los resultados comparativos son los siguientes

	PGD α_1	**ADMM** α_1	**ADMM** α_2
x^*	$(-0.1327, 0.1270)^T$	$(-0.1327, 0.1269)^T$	$(-0.1327, 0.1272)^T$
N iteraciones	35	40	9
Tiempo	$2.982ms$	$3.445ms$	$0.715ms$

Se observa que *PGD* insume un tiempo similar a *ADMM* con α_1. Mientras que cuando se escoge α_2 en *ADMM* se tiene que este es x4.17 veces más rápido que *PGD*.
Lo anterior indica que frente una **correcta elección del paso**, se puede lograr que la velocidad de *ADMM* se bastante superior a *PGD*.

Nota: la comparación de tiempos esta muy sujeta a la implementación de cada método, por lo tanto se busco optimizar ambos métodos lo más posible. A modo de ejemplo, en *ADMM* el calculo de la matriz inversa se computo una única vez.

A continuación se presenta como evoluciona la función objetivo con las iteraciones para los tres casos de estudio.

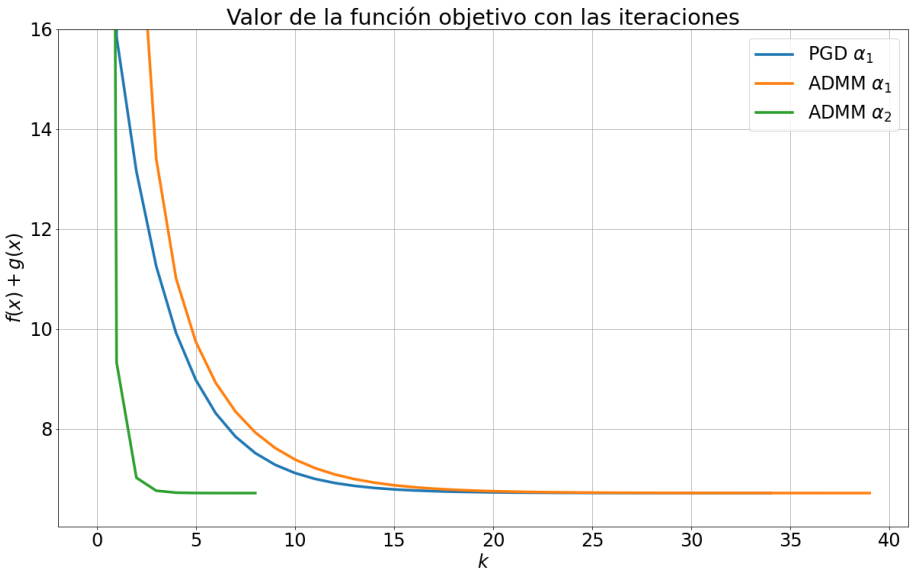

Figure 1

La gráfica anterior confirma los resultados numéricos obtenidos. La curva verde logra llegar a un valor cercano al óptimo en menos iteraciones que la azul y la naranja.

En conclusión, si se elige correctamente el paso, utilizar $ADMM$ seria más ventajoso en cuanto al tiempo y cantidad de iteraciones.

Código

El código se encuentra adjunto en el archivo *code.zip*. Este zip esta compuesto por:

- *utils.py*: Tiene definidas las clases útiles para la resolución de problema.
- *ej2.py*: Utiliza las clases del *utils.py* para resolver el problema con $ADMM$ y PGD. Además repite la resolución 10000 para poder calcular los tiempos insumidos. También se encarga de mostrar los resultados y gráficas adecuadas.