Gonghan Xu, SDS 631, hw5

1. (BV Exercise 9.7) (a) ∵ Δx_{nsd} is a descent direction ∴ $\nabla f(x)^T \Delta x_{nsd} < 0$ ①

By definition of Δx_{nsd}, we have $\Delta x_{nsd} \equiv \operatorname{argmin}\{\nabla f(x)^T v \mid \|v\| \le 1\}$ ②

∴ Combining ①, ②, we have $\nabla f(x)^T \Delta x_{nsd} = \min\{\nabla f(x)^T v \mid \|v\| \le 1\}$ (then use ①)

$= -\max\{-\nabla f(x)^T v \mid \|v\| \le 1\} = -\sup\{\nabla f(x)^T \cdot (-v) \mid \|-v\| \le 1\}$ (because $\{v \mid \|v\| \le 1\}$ is a compact set so max = sup)

$= -\sup\{\nabla f(x)^T v' \mid \|v'\| \le 1\}$ (where we have set $v' \equiv -v$)

$= -\sup\{\langle \nabla f(x), v' \rangle \mid \|v'\| \le 1\} = -\|\nabla f(x)\|_*$ ③, where we have used the definition of $\|\cdot\|_*$

(b) $\nabla f(x)^T \Delta x_{sd} = \nabla f(x)^T \cdot (\|\nabla f(x)\|_* \Delta x_{nsd}) = \|\nabla f(x)\|_* (\nabla f(x)^T \Delta x_{nsd})$ (then use 1.(a) ③)

$= \|\nabla f(x)\|_* \{-\|\nabla f(x)\|_*\} = -\|\nabla f(x)\|_*^2$ ①

(c) Now we try to find $v \in \mathbb{R}^n$ that minimize $\nabla f(x)^T v + \frac{1}{2}\|v\|^2$. ①

Note that $v = \|v\| \cdot \frac{v}{\|v\|} \equiv \|v\| \cdot \hat{v}$ ②, where we define $\hat{v} \equiv \frac{v}{\|v\|}$ ③ Then $\|\hat{v}\| = 1$ ④

Then our objective function $h(v) \equiv \nabla f(x)^T v + \frac{1}{2}\|v\|^2 = \frac{1}{2}\|v\|^2 + \|v\| \nabla f(x)^T \hat{v}$ ⑤

Note that $\|v\|$ and $\hat{v}$ are independent on each other. So we can minimize $h(v)$ with respect to them in a separate-two-step manner. First, with $\|v\|$ fixed, since $\|v\| \ge 0$ then we want to minimize $\nabla f(x)^T \hat{v}$ in order to minimize $h(v)$. Then $\hat{v}^* = \operatorname{argmin}\{\nabla f(x)^T \hat{v} \mid \|\hat{v}\| = 1\} = \Delta x_{nsd}$ ⑥

Note that when $\hat{v} = \hat{v}^* = \Delta x_{nsd}$, $\nabla f(x)^T \hat{v}^* = \nabla f(x)^T \Delta x_{nsd} \le 0$ ⑦

Then using the quadratic polynomial minimization property $-\frac{b}{2a} = \operatorname{argmin}_x\{ax^2+bx+c \mid a \ge 0\}$, we have: $\|v\|^* = \operatorname{argmin}\{\frac{1}{2}\|v\|^2 + \|v\| \nabla f(x)^T \Delta x_{nsd} \mid \|v\| \ge 0\} = -\nabla f(x)^T \Delta x_{nsd} \ge 0$ ⑧

∴ $\|v\|^* = -\nabla f(x)^T \Delta x_{nsd}$ ⑨ and $\hat{v}^* = \Delta x_{nsd}$ ⑩

Then using 1.(a) ③, we have $\|v\|^* = -\{\nabla f(x)^T \Delta x_{nsd}\} = \|\nabla f(x)\|_*$ ⑪

∴ $v^* = \|v\|^* \cdot \hat{v}^* = \|\nabla f(x)\|_* \Delta x_{nsd} = \Delta x_{sd}$ ⑫ (where we used the definition of Δx_{sd})

∴ $\Delta x_{sd} = v^* \equiv \operatorname{argmin}_v\{\nabla f(x)^T v + \frac{1}{2}\|v\|^2\}$ ⑬

2. (AE Exercise 9.1(b)) $\because \gamma > 1$, $\therefore x^{(0)} \equiv (\gamma, 1) \in \{(x_1, x_2) \mid |x_2| \le x_1\}$ ①

Now we want to show $x_1^{(k)} = \gamma\left(\frac{\gamma-1}{\gamma+1}\right)^k$, $x_2^{(k)} = \left(\frac{1-\gamma}{\gamma+1}\right)^k$ ②. by mathematical induction. Note that $x_1^{(k)} > |x_2^{(k)}|$ ③ since $\gamma > 1$.

Define $D_1 \equiv \{(x_1, x_2)^T \mid |x_2| \le x_1\}$ ④. Then when $x \equiv (x_1, x_2)^T \in D_1$, we have: $\frac{\partial f}{\partial x_1} = \frac{1}{2}\{x_1^2 + \gamma x_2^2\}^{-\frac{1}{2}} \cdot 2x_1 = \frac{1}{f(x)} \cdot x_1$ ⑤

$\frac{\partial f}{\partial x_2} = \frac{1}{2}\{x_1^2 + \gamma x_2^2\}^{-\frac{1}{2}} \cdot 2\gamma \cdot x_2 = \frac{1}{f(x)} \cdot \gamma x_2$ ⑥, $\nabla f(x) = \frac{1}{f(x)} \cdot (x_1, \gamma x_2)^T$ ⑥

Now we use mathematical induction:

(i). The base case: when $k=0$, we have $x_1^{(0)} = \gamma\left(\frac{\gamma-1}{\gamma+1}\right)^0 = \gamma$ ⑦

and $x_2^{(0)} = \left(\frac{1-\gamma}{1+\gamma}\right)^0 = 1$ ⑧ This is indeed our starting point.

(ii). Suppose the k-th step iterates satisfy $x_1^{(k)} = \gamma\left(\frac{\gamma-1}{\gamma+1}\right)^k$ ⑨ and $x_2^{(k)} = \left(\frac{1-\gamma}{1+\gamma}\right)^k$ ⑩, then we want to show $x_1^{(k+1)} = \gamma\left(\frac{\gamma-1}{\gamma+1}\right)^{k+1}$ and $x_2^{(k+1)} = \left(\frac{1-\gamma}{1+\gamma}\right)^{k+1}$ ⑪, where $k \in N$. Now given ⑨, ⑩, we have

$$\tilde{x}^{(k+1)}(t) \equiv x^{(k)} - t\nabla f(x^{(k)}) = \left(\gamma\left(\frac{\gamma-1}{\gamma+1}\right)^k, \left(\frac{1-\gamma}{1+\gamma}\right)^k\right)^T - t \cdot \frac{1}{f(x^{(k)})}\left(\gamma\left(\frac{\gamma-1}{\gamma+1}\right)^k, \gamma\left(\frac{1-\gamma}{1+\gamma}\right)^k\right)^T$$

$$= \left(\left(1 - \frac{t}{f(x^{(k)})}\right) \cdot \gamma\left(\frac{\gamma-1}{\gamma+1}\right)^k, \left(1 - \frac{t\gamma}{f(x^{(k)})}\right) \cdot \left(\frac{1-\gamma}{1+\gamma}\right)^k\right)^T$$ ⑫

Then $\hat{f}(t) \equiv f(\tilde{x}^{(k+1)}(t)) = \left\{\left(1 - \frac{t}{f(x^{(k)})}\right)^2 \gamma^2 \left(\frac{\gamma-1}{\gamma+1}\right)^{2k} + \gamma\left(1 - \frac{t\gamma}{f(x^{(k)})}\right)^2 \left(\frac{1-\gamma}{1+\gamma}\right)^{2k}\right\}^{\frac{1}{2}}$

$$= \left\{\left[\gamma^2\left(\frac{\gamma-1}{\gamma+1}\right)^{2k} + \gamma^3\left(\frac{1-\gamma}{1+\gamma}\right)^{2k}\right]\frac{t^2}{[f(x^{(k)})]^2} - \frac{2t}{f(x^{(k)})}\left[\gamma^2\left(\frac{\gamma-1}{\gamma+1}\right)^{2k} + \gamma^2\left(\frac{1-\gamma}{1+\gamma}\right)^{2k}\right] + h(\gamma)\right\}^{\frac{1}{2}}$$ ⑬

where $h(\gamma)$ is a function of γ only

Then to minimize the quadratic expression inside $\{\}^{\frac{1}{2}}$ in ⑬ of t, we only need to take

$$t^* = \frac{2\left[\gamma^2\left(\frac{\gamma-1}{\gamma+1}\right)^{2k} + \gamma^2\left(\frac{1-\gamma}{1+\gamma}\right)^{2k}\right]}{2\left[\gamma^2\left(\frac{\gamma-1}{\gamma+1}\right)^{2k} + \gamma^3\left(\frac{1-\gamma}{1+\gamma}\right)^{2k}\right]} \cdot f(x^{(k)})$$ (where we used $x^* = -\frac{b}{2a} = \arg\min_x\{ax^2+bx+c\}$ for $a>0$)

$= \left(\frac{2}{1+\gamma}\right) \cdot f(x^{(k)})$ ⑭ Then plugging $t = t^*$ into ⑫, we get:

$x_1^{(k+1)} = \left(1 - \frac{2}{1+\gamma}\right) \cdot \gamma \cdot \left(\frac{\gamma-1}{\gamma+1}\right)^k = \gamma\left(\frac{\gamma-1}{\gamma+1}\right)^{k+1}$ ⑮

$x_2^{(k+1)} = \left(1 - \gamma \cdot \frac{2}{1+\gamma}\right) \cdot \left(\frac{1-\gamma}{1+\gamma}\right)^k = \left(\frac{1-\gamma}{1+\gamma}\right)^{k+1}$ ⑯

$\therefore$ We can see that ⑮ and ⑯ do match our target form ⑪.

$\therefore$ We have proven the k-th step iterates $x_1^{(k)}$ and $x_2^{(k)}$ satisfy the closed form ② by mathematical induction.

3. (Proximal Gradient Descent I)

(a) We have $prox_{\alpha,h}(x-\alpha\cdot\nabla g(x)) \equiv \underset{y}{argmin}\left\{\frac{1}{2\alpha}\|x-\alpha\nabla g(x)-y\|_2^2 + h(y)\right\}$

$= \underset{y}{argmin}\left\{\frac{1}{2\alpha}\|(y-x)+\alpha\nabla g(x)\|_2^2 + h(y)\right\} = \underset{y}{argmin}\left\{\frac{1}{2\alpha}[(y-x)+\alpha\nabla g(x)]^T[(y-x)+\alpha\nabla g(x)] + h(y)\right\}$

$= \underset{y}{argmin}\left\{\frac{1}{2\alpha}\|y-x\|_2^2 + \nabla g(x)^T(y-x) + \frac{\alpha}{2}\|\nabla g(x)\|_2^2 + h(y)\right\}$

$= \underset{y}{argmin}\left\{U(x,y) + \left[\frac{\alpha}{2}\|\nabla g(x)\|_2^2 - g(x)\right]\right\}$ (Then since $\left[\frac{\alpha}{2}\|\nabla g(x)\|_2^2 - g(x)\right]$ does not depend on y, we can drop it out of the $\underset{y}{argmin}\{\cdots\}$)

$= \underset{y}{argmin}\{U(x,y)\}$ ①

(b) $\underset{v\in C}{argmin}\left\{g(x) + \langle\nabla g(x), v-x\rangle + \frac{1}{2\alpha}\|v-x\|_2^2\right\}$

$= \underset{v\in R^n}{argmin}\left\{g(x) + \langle\nabla g(x), v-x\rangle + \frac{1}{2\alpha}\|v-x\|_2^2 + h(v)\right\}$ where $h(v) \equiv \begin{cases}\infty & \text{if } v\notin C \\ 0 & \text{if } v\in C\end{cases}$ → ①

(then using the result from part (a))

$= prox_{\alpha,h}(x-\alpha\nabla g(x)) \equiv \underset{v\in R^n}{argmin}\left\{\frac{1}{2\alpha}\|(x-\alpha\nabla g(x))-v\|_2^2 + h(v)\right\}$ (then use definition ①)

$= \underset{v\in C}{argmin}\left\{\frac{1}{2\alpha}\|(x-\alpha\nabla g(x))-v\|_2^2\right\} = \underset{v\in C}{argmin}\left\{\|[x-\alpha\nabla g(x)]-v\|_2^2\right\}$

$= \Pi_C(x-\alpha\nabla g(x))$ ② because $\Pi_C(u) \equiv \underset{v\in C}{argmin}\left\{\|u-v\|_2^2\right\}$ ③

$= y$ ④

(c) Since $g(x)$ is M-smooth, i.e. $\|\nabla g(x)-\nabla g(y)\|_2 \le M\|x-y\|_2$ for $\forall x, y\in R^n$, then from lecture notes 16 page 5, we have proven the result:

$|g(y)-[g(x)+\nabla g(x)^T(y-x)]| \le \frac{M}{2}\|x-y\|_2^2$ ①

$\therefore\ g(y)-[g(x)+\nabla g(x)^T(y-x)] \le \frac{M}{2}\|x-y\|_2^2$

$\therefore\ g(y) \le g(x)+\nabla g(x)^T(y-x)+\frac{M}{2}\|x-y\|_2^2$ (then use $M\le\frac{1}{\alpha}$)

$\le g(x)+\nabla g(x)^T(y-x)+\frac{M}{2}\|y-x\|_2^2$ ② $= U(x,y)-h(y)$ ③

$\therefore\ g(y) \le U(x,y)-h(y)$

$\therefore\ U(x,y) \ge g(y)+h(y) = f(y)$ ④

(d) For $\forall x^*$ such that $G_\alpha(x^*)=0$, we have $x^* = prox_{\alpha,h}(x^* - \alpha\nabla g(x^*))$ ①

$\equiv \underset{y}{argmin}\left\{\frac{1}{2\alpha}\|x^* - \alpha\nabla g(x^*) - y\|_2^2 + h(y)\right\}$ ②.

Now define $p(y) \equiv \frac{1}{2\alpha}\|x^* - \alpha\nabla g(x^*) - y\|_2^2 + h(y)$

$= \frac{1}{2\alpha}(x^* - \alpha\nabla g(x^*) - y)^T(x^* - \alpha\nabla g(x^*) - y) + h(y)$

$= \frac{1}{2\alpha}\left\{y^Ty - 2(x^* - \alpha\nabla g(x^*))^Ty + \|x^* - \alpha\nabla g(x^*)\|_2^2\right\} + h(y)$ ③

$\therefore x^* = \underset{y}{argmin}\{p(y)\}$ ④ Note that $p(y)$ is a differentiable function since $h(y)$ is differentiable. Then using the first-order optimality condition on ④, we have $\langle \nabla p(y)|_{y=x^*}, y - x^* \rangle \geq 0$ for $\forall y \in R^n$ ⑤

Note that $\nabla p(y)|_{y=x^*} = \left[\frac{1}{2\alpha}\left\{2y - 2(x^* - \alpha\nabla g(x^*))\right\} + \nabla h(y)\right]\Big|_{y=x^*}$ (from ③)

$= \frac{1}{2\alpha}\left\{2x^* - 2x^* + 2\alpha\nabla g(x^*)\right\} + \nabla h(x^*)$

$= \nabla g(x^*) + \nabla h(x^*) = \nabla f(x^*)$ ⑥ (since $f(x) \equiv g(x) + h(x)$)

Then plugging ⑥ in ⑤, we have $\langle \nabla f(x^*), y - x^* \rangle \geq 0$ ⑦ for $\forall y \in R^n$. Then since $y \in R^n$, we have $y - x^* \in R^n$ can be any arbitrary vector in R^n. $\therefore$ For ⑦ to hold for $\forall y \in R^n$, we must have $\nabla f(x^*) = 0$ ⑧. $\therefore$ We have proven for $\forall x^*$ such that $G_\alpha(x^*) = 0$, we must have $\nabla f(x^*) = 0$, i.e. x^* is a stationary point of f.

4. (a) We can simply redo the entire procedure as in problem 3 part (d), except that 3(d) ① becomes $x^{k+1} = prox_{\alpha,h}\{x^k - \alpha\nabla g(x^k)\}$ ①

Then 3(d) ⑥ becomes: $\nabla p(y)|_{y=x^{k+1}} = \frac{1}{2\alpha}\left\{2x^{k+1} - 2x^k + 2\alpha\nabla g(x^k)\right\} + \nabla h(x^{k+1})$ ②

Then using the same reasoning as in 3(d), we have:

$\nabla p(x^{k+1}) = \frac{1}{\alpha}\left\{x^{k+1} - x^k + \alpha\nabla g(x^k)\right\} + \nabla h(x^{k+1}) = 0$ ③

$\therefore$ Equivalently, we have $\nabla g(x^k) + \frac{1}{\alpha}(x^{k+1} - x^k) + \nabla h(x^{k+1}) = 0$ ④

$\therefore \nabla g(x^k) - G_\alpha(x^k) + \nabla h(x^{k+1}) = 0$ ⑤

Note that ① trivially comes from $x^{k+1} = x^k - \alpha G_\alpha(x^k)$

$= x^k - \alpha\cdot\frac{1}{\alpha}\left\{x^k - prox_{\alpha,h}(x^k - \alpha\nabla g(x^k))\right\}$

$= prox_{\alpha,h}(x^k - \alpha\nabla g(x^k))$ ⑥

(b) ∵ $h(x)$ is convex and differentiable, ∴ By first-order condition for convex functions, we have $h(x^k) \geq h(x^{k+1}) + \langle \nabla h(x^{k+1}), x^k - x^{k+1} \rangle$ ①

And by definition of x^{k+1} for proximal gradient updates, we have:

$x^k - x^{k+1} = \alpha G_\alpha(x^k)$ ② (see definition (5) in the prompt) Then plugging ② into ①, we get: $h(x^{k+1}) + \alpha \langle \nabla h(x^{k+1}), G_\alpha(x^k) \rangle \leq h(x^k)$ ③

(c) First we prove $f(x^{k+1}) \leq U(x^k, x^{k+1})$ ① This is very straightforward. We can simply plug $x = x^k$ and $y = x^{k+1}$ into 3(d) ④, then we directly get: $U(x^k, x^{k+1}) \geq g(x^{k+1}) + h(x^{k+1}) = f(x^{k+1})$ ②

Next, we prove $U(x^k, x^{k+1}) \leq f(x^k) - \frac{\alpha}{2} \| G_\alpha(x^k) \|_2^2$ ③

Use "$\Leftrightarrow$" to denote equivalence. Then using the definition of $U(x,y)$,

③ $\Leftrightarrow g(x^k) + \langle \nabla g(x^k), x^{k+1} - x^k \rangle + \frac{1}{2\alpha} \| x^{k+1} - x^k \|_2^2 + h(x^{k+1}) \leq f(x^k) - \frac{\alpha}{2} \| G_\alpha(x^k) \|_2^2$

$\Leftrightarrow g(x^k) - \alpha \langle \nabla g(x^k), G_\alpha(x^k) \rangle + \frac{1}{2\alpha} \| \alpha^2 G_\alpha(x^k) \|_2^2 + h(x^{k+1}) \leq g(x^k) + h(x^k) - \frac{\alpha}{2} \| G_\alpha(x^k) \|_2^2$

(where we have used $x^{k+1} = x^k - \alpha G_\alpha(x^k)$ and $f = g + h$)

$\Leftrightarrow -\alpha \langle \nabla g(x^k), G_\alpha(x^k) \rangle + \alpha \| G_\alpha(x^k) \|_2^2 + h(x^{k+1}) \leq h(x^k)$ ④

$\Leftrightarrow -\alpha \langle \nabla g(x^k), G_\alpha(x^k) \rangle + \alpha \langle G_\alpha(x^k), G_\alpha(x^k) \rangle + h(x^{k+1}) \leq h(x^k)$

$\Leftrightarrow \alpha \langle [G_\alpha(x^k) - \nabla g(x^k)], G_\alpha(x^k) \rangle + h(x^{k+1}) \leq h(x^k)$ ⑤

(Then plugging $\nabla h(x^{k+1}) = G_\alpha(x^k) - \nabla g(x^k)$ from 4.(a) ⑤ into ⑤)

$\Leftrightarrow \alpha \langle \nabla h(x^{k+1}), G_\alpha(x^k) \rangle + h(x^{k+1}) \leq h(x^k)$ ⑥

Notice that inequality ⑥ is exactly the result of part (b) that we have proven. Then by equivalence, we can immediately see that ③ is true. Then combining ② and ③, we have: $f(x^{k+1}) \leq U(x^k, x^{k+1}) \leq f(x^k) - \frac{\alpha}{2} \| G_\alpha(x^k) \|_2^2$ ⑦

5. (a) $\text{prox}_h(x) \equiv \underset{y}{\text{argmin}} \left\{ \frac{1}{2}\|x-y\|_2^2 + h(y) \right\}$ ①

Then define $p(y) \equiv \frac{1}{2}\|x-y\|_2^2 + h(y)$ ②, we have $\nabla p(y) = \nabla_y \left\{ \frac{1}{2}(y-x)^T(y-x) + h(y) \right\}$

$= \nabla_y \left\{ \frac{1}{2}y^Ty - x^Ty + \frac{1}{2}x^Tx + h(y) \right\} = y - x + \nabla h(y)$ ③

Then similar to what we did in problem 3(d), by optimality condition, we have $\nabla p(y)\big|_{y=\text{prox}_h(x)} = 0$ ④ ∴ Plugging ③ into ④, we have:

$\text{prox}_h(x) - x + \nabla h(\text{prox}_h(x)) = 0$ ⑤ ∴ $\nabla h(\text{prox}_h(x)) = x - \text{prox}_h(x)$ ⑥

Now using the result of BV Exercise 3.11, which is homework 2 Problem b, we know that $\langle \nabla h(x) - \nabla h(y), x-y \rangle \geq 0$ ⑦ for $\forall x, y \in \mathbb{R}^n$ since h is a differentiable convex function. Then plugging $x = \text{prox}_h(x_1)$, $y = \text{prox}_h(x_2)$ into ⑦, we have: $\langle \nabla h(\text{prox}_h(x_1)) - \nabla h(\text{prox}_h(x_2)), \text{prox}_h(x_1) - \text{prox}_h(x_2) \rangle \geq 0$ ⑧ Then plugging ⑥ into ⑧, we have: $\langle [x_1 - \text{prox}_h(x_1)] - [x_2 - \text{prox}_h(x_2)], \text{prox}_h(x_1) - \text{prox}_h(x_2) \rangle \geq 0$ ⑨

∴ $\langle (x_1 - x_2) - [\text{prox}_h(x_1) - \text{prox}_h(x_2)], \text{prox}_h(x_1) - \text{prox}_h(x_2) \rangle \geq 0$

∴ $\langle x_1 - x_2, \text{prox}_h(x_1) - \text{prox}_h(x_2) \rangle \geq \langle \text{prox}_h(x_1) - \text{prox}_h(x_2), \text{prox}_h(x_1) - \text{prox}_h(x_2) \rangle$

∴ $\langle \text{prox}_h(x_1) - \text{prox}_h(x_2), x_1 - x_2 \rangle \geq \|\text{prox}_h(x_1) - \text{prox}_h(x_2)\|_2^2$ ⑩

(b) Using Cauchy-Schwarz inequality, we have:

$\langle \text{prox}_h(x_1) - \text{prox}_h(x_2), x_1 - x_2 \rangle \leq \|\text{prox}_h(x_1) - \text{prox}_h(x_2)\|_2 \cdot \|x_1 - x_2\|_2$ ①

Then combining ① and 5.(a) ⑩, we have:

$\|\text{prox}_h(x_1) - \text{prox}_h(x_2)\|_2 \cdot \|x_1 - x_2\|_2 \geq \|\text{prox}_h(x_1) - \text{prox}_h(x_2)\|_2^2$ ②

∴ $\|x_1 - x_2\|_2 \geq \|\text{prox}_h(x_1) - \text{prox}_h(x_2)\|_2$ ③

Note that if $\|\text{prox}_h(x_1) - \text{prox}_h(x_2)\|_2 = 0$, ③ will trivially hold. If $\|\text{prox}_h(x_1) - \text{prox}_h(x_2)\|_2 \neq 0$, then we can simply divide both sides of ② by this term to get ③.

6. (a) $V' \equiv \Pi_C(V) \equiv \underset{u \in C}{\operatorname{argmin}} \|V-u\|_2 = \underset{u \in C}{\operatorname{argmin}} \|V-u\|_2^2$ ①

Then if $s<n$, we know for $u \in C \equiv \{x \mid \|x\|_0 \le s\}$, there must be at least $(n-s)$ zero entries in u. Then for $\forall\, u \in C$, let set J contains the indices of $(n-s)$ zero entries in u. If there are more than $(n-s)$ zero entries in u, we can pick the indices of $(n-s)$ of them arbitrarily.

Then $\|V-u\|_2^2 = \sum_{i=1}^{n} (V_i-u_i)^2 \ge \sum_{i \in J} (V_i-0)^2 = \sum_{i \in J} |V_i|^2 \ge \sum_{i \notin V} |V_i|^2$ ②

where the last inequality comes from the definition of V such that V contains the indices of the largest s entries in absolute value of V. And notice that when $u \in C$ satisfies $u_j = \begin{cases} V_j, & \text{if } j \in V \\ 0, & \text{if } j \notin V \end{cases}$ ③, we have $\|V-u\|_2^2 = \sum_{i \notin V} |V_i|^2$ ④

Then combining ② and ③, ④, we know that

$\min_{u \in C} \|V-u\|_2^2 = \sum_{i \notin V} |V_i|^2$ ⑤, and $\Pi_C(V) \equiv \underset{u \in C}{\operatorname{argmin}} \|V-u\|_2^2 = V'$ such that

$V'_j = \begin{cases} V_j, & \text{if } j \in V \\ 0, & \text{if } j \notin V \end{cases}$ ⑥

If $s \ge n$, then $V' \equiv \Pi_C(V) = V$ ⑦ because $V \in C \equiv \{x \mid \|x\|_0 \le s\}$ and $\|V-V\|_2^2 = 0$, which is the least possible value for L_2 norm. In this case, since $s \ge n$, we have $V = \{1, 2, 3, \ldots, n\}$. Then ⑥ will still hold, trivially.

(b) See notebook implementation (attached in next page)

(c) See notebook output cells (attached in next page)

Generate data from sparse linear regression model

In [2]:
```python
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(431)

def generate_data(m, n, s, sigma = 0.1):
    # Step 1: Generate X from Gaussian distribution
    X = np.random.randn(m, n)

    # Step 2: Create beta_star with first s entries as 1 and the rest 0
    beta_star = np.zeros(n)
    beta_star[:s] = 1

    y = X @ beta_star + sigma * np.random.randn(m,)

    return X, y, beta_star


# evaluate the error
def l2_error(beta, beta_star):
    return np.linalg.norm(beta - beta_star)


# # Example usage:
# m, n, s = 100, 10, 3  # m samples, n features, s sparsity
# X, y, beta_star = generate_data(m, n, s)
# X.shape, y.shape, beta_star.shape  # Checking the shapes of the generated data



def plot_results(iterates, objective_values, beta_star):
    # Calculate the errors between iterates and beta_star
    errors = [l2_error(beta_t, beta_star) for beta_t in iterates]

    # Create Figure 1: Error vs. Iteration
    plt.figure(figsize=(10, 4))
    plt.subplot(1, 2, 1)
    plt.plot(errors)
    plt.xlabel('Iteration')
    plt.ylabel('Error (||beta_t - beta_star||)')
    plt.title('Error vs. Iteration')

    # Create Figure 2: Function Value vs. Iteration
    plt.subplot(1, 2, 2)
    plt.plot(objective_values)
    plt.xlabel('Iteration')
    plt.ylabel('Function Value')
    plt.title('Function Value vs. Iteration')

    plt.tight_layout()
    plt.show()
```

Generate 200 data points with dimension $n = 100$ and sparsity is $s = 5$.

In [3]:
```python
m, n, s = 200, 100, 5  # m samples, n features, s sparsity
X, y, beta_star = generate_data(m, n, s)
```

Method 1: Convex optimization via CVX

We use CVXPY to solve two convex optimization problems:

$\min \ell(\beta) + \lambda \cdot \|\beta\|_1$,

$\min \ell(\beta)$ subject to $\|\beta\|_1 \leq \lambda$.

In [6]:
```
import cvxpy as cp

def solve_cvx_reg(lambda_val, X, y):
    m, n = X.shape

    beta_cvx = cp.Variable(n)

    least_squares = cp.norm2(y - X @ beta_cvx)**2

    problem = cp.Problem(cp.Minimize(least_squares + lambda_val * cp.norm(beta_cvx, 1)))

    problem.solve()

    return beta_cvx.value

def solve_cvx_constraint(lambda_val, X, y):
    m, n = X.shape

    beta_cvx = cp.Variable(n)

    least_squares = cp.norm2(y - X @ beta_cvx)**2

    objective = cp.Minimize(cp.norm2(y - X @ beta_cvx)**2)
    constraints = [cp.norm(beta_cvx, 1) <= lambda_val]
    problem = cp.Problem(objective, constraints)
    problem.solve()
    return beta_cvx.value

beta_cvx_con = solve_cvx_constraint(5, X, y)
beta_cvx_reg = solve_cvx_reg(6, X, y)

print(f"L1-regularized problem -- Error of Convex optimization from CVXPY: \n{l2_error(b
print(f"L1-constrained problem -- Error of Convex optimization from CVXPY: \n{l2_error(b

if max(l2_error(beta_cvx_reg, beta_star), l2_error(beta_cvx_con, beta_star)) < 0.05:
  print("\nThese two methods recovers the parameter acurrately \n")
else:
  print("\nThe error seems a bit large, perhaps you need to check the code or tune the h
```

```
L1-regularized problem -- Error of Convex optimization from CVXPY:
0.0319505909425948

L1-constrained problem -- Error of Convex optimization from CVXPY:
0.02642879312111666

These two methods recovers the parameter acurrately
```

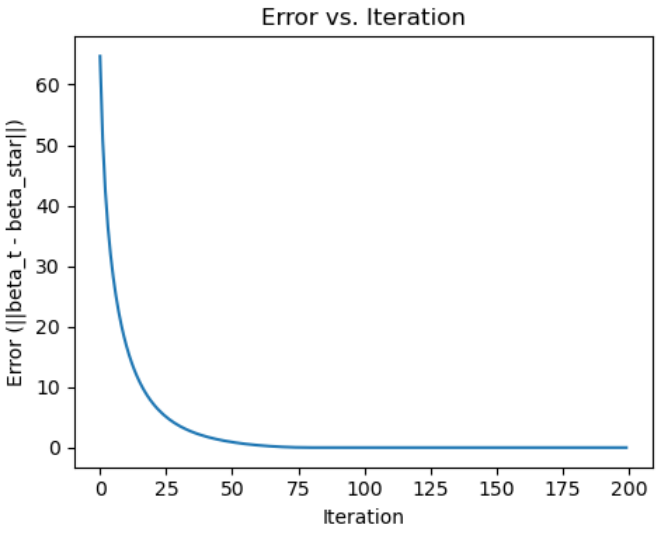
Method 2: Proximal Gradient Descent

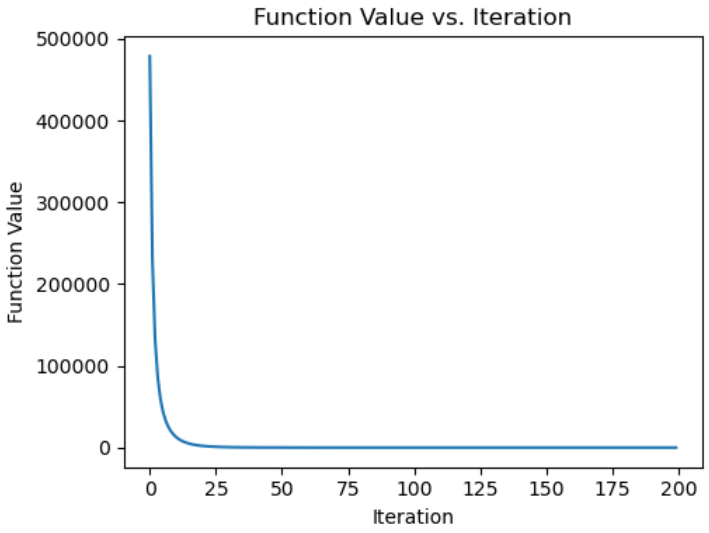
The objective function is $\ell(\beta) + \lambda\|\beta\|_1$. The proximal operator reduces to soft-thresholding.

In [9]:
```
# v is the main argument, and param is the thresholding parameter
```

```python
def soft_threshold(v, param):
    # Soft_thresholding function
    # Your code here
    return np.sign(v) * np.maximum(np.abs(v) - param, 0)


def proximal_gradient(beta_0, X, y, alpha, lambda_, N_iter):
    # Initialize variables
    beta = beta_0
    objective_values = []  # To store the sequence of objective values
    iterates = []  # To store the sequence of iterates (beta values)

    for iteration in range(N_iter):
        # Compute the gradient of the objective function
        gradient = -2 * X.T.dot(y - X.dot(beta))

        # Update beta using soft-thresholding
        beta = soft_threshold(beta - alpha * gradient, alpha * lambda_)

        # Calculate the objective value and append to the list
        objective_value = np.linalg.norm(y - X.dot(beta))**2 + lambda_ * np.linalg.norm(
        objective_values.append(objective_value)

        # Append the current iterate (beta) to the list
        iterates.append(beta)

    return beta, iterates, objective_values


## Now let's test proximal gradient

# initialization
beta_init = 10 * np.ones(n)
lambda_prox = 3
stepsize = 0.001
N_iter = 200

beta_prox, beta_prox_seq, fun_prox_seq = proximal_gradient(beta_init, X, y, stepsize, la

print(f"L1-regularized problem -- Error of Proximal gradient: \n{l2_error(beta_prox, bet

if l2_error(beta_prox, beta_star) < 0.05:
  print("\nProximal Gradient recovers the parameter acurrately \n")
else:
  print("\nThe error seems a bit large, perhaps you need to check the code or tune the h

# generate plots
plot_results(beta_prox_seq, fun_prox_seq, beta_star)
```

```
L1-regularized problem -- Error of Proximal gradient:
0.027332228507959237


Proximal Gradient recovers the parameter acurrately
```





Method 3: Projected Gradient Descent

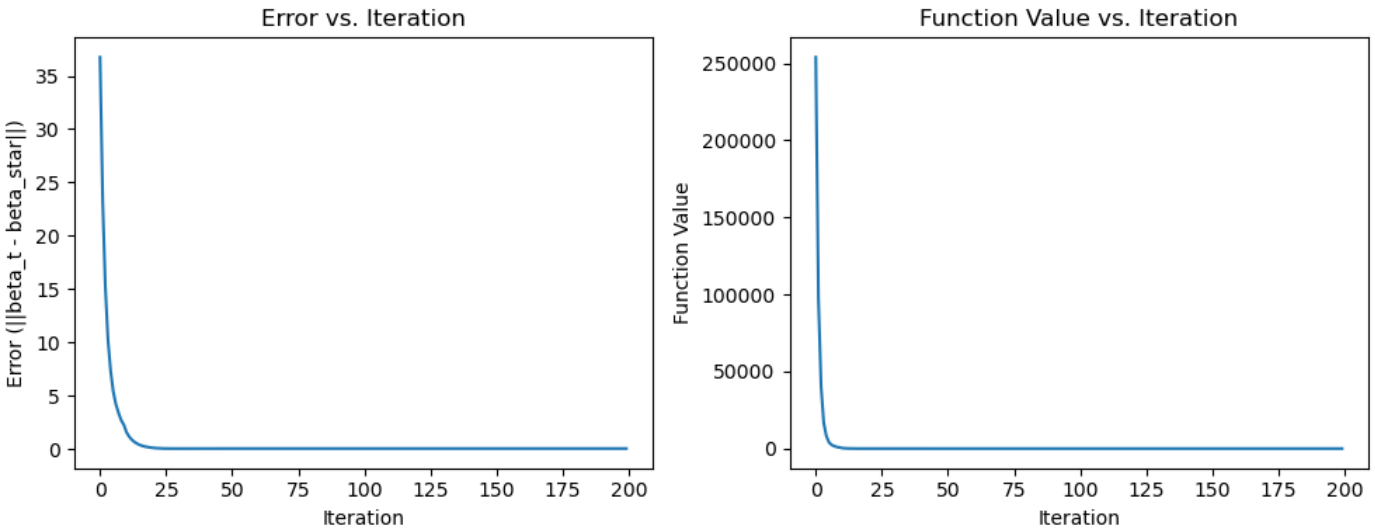
The optimization problem is $\min \ell(\beta)$ subject to $\|\beta\|_0 \leq t$ for some integer t.

In [10]:

```python
def project_largest_t_elements(beta, t):
    # Your code here
    n = len(beta)
    if t < n:
        # Get the indices of the (n-t) smallest entries in absolue value
        # The entries of the projection vector will be 0 at these indices.
        indices = np.argsort(np.abs(beta))[:n-t]
        beta = beta.copy()
        beta[indices] = 0

    return beta


def projected_gradient(beta_0, X, y, alpha, t, N_iter):
    # Initialize variables
    beta = beta_0
    objective_values = []  # To store the sequence of objective values
    iterates = []  # To store the sequence of iterates (beta values)

    for iteration in range(N_iter):
        # Compute the gradient of the objective function
        gradient = -2 * X.T.dot(y - X.dot(beta))

        # Update beta using projection of (beta - alpha * gradient)
        beta = project_largest_t_elements(beta - alpha * gradient, t)
        # print(beta)
        # Calculate the objective value and append to the list
        objective_value = np.linalg.norm(y - X.dot(beta))**2
        objective_values.append(objective_value)

        # Append the current iterate (beta) to the list
        iterates.append(beta)

    return beta, iterates, objective_values


## Now let's test proximal gradient

# initialization
beta_init = 10 * np.ones(n)
```

```
t = 10
stepsize = 0.001
N_iter = 200

beta_proj, beta_proj_seq, fun_proj_seq = projected_gradient(beta_init, X, y, stepsize, t

print(f"L0-constrained problem -- Error of Projected gradient: \n{l2_error(beta_proj, be

if l2_error(beta_proj, beta_star) < 0.05:
  print("\nProjected Gradient recovers the parameter acurrately \n")
else:
  print("\nThe error seems a bit large, perhaps you need to check the code or tune the h

# generate plots
plot_results(beta_proj_seq, fun_proj_seq, beta_star)
```

```
L0-constrained problem -- Error of Projected gradient:
0.035612466947055346


Projected Gradient recovers the parameter acurrately
```



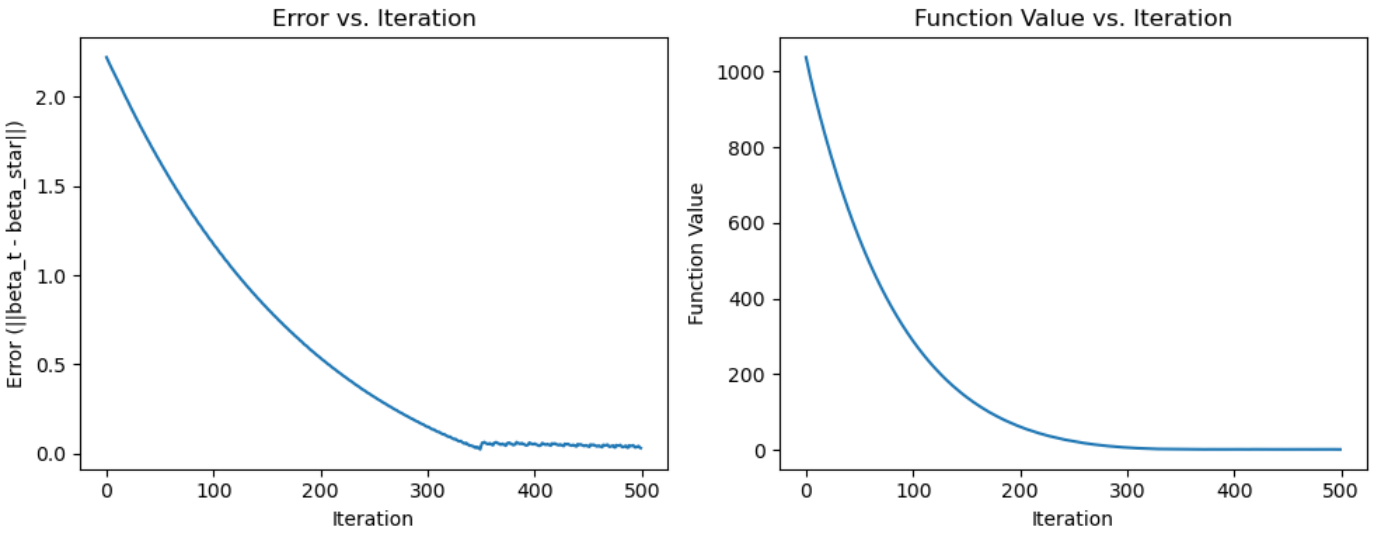
Method 4: Frank-Wolfe Method

The optimization problem is $\min \ell(\beta)$ subject to $\|\beta\|_1 \leq \lambda$.

In [12]:
```
def update_direction(grad, lambda_):
    # Your code here
    direc = cp.Variable(len(grad))
    objective = cp.Minimize(direc @ grad)
    constraints = [cp.norm(direc, 1) <= lambda_]
    problem = cp.Problem(objective, constraints)
    problem.solve()
    return direc.value


def frank_wolfe(beta_0, X, y, alpha, lambda_, N_iter):
    # Initialize variables
    beta = beta_0
    objective_values = []  # To store the sequence of objective values
    iterates = []  # To store the sequence of iterates (beta values)

    for iteration in range(N_iter):
        # Compute the gradient of the objective function
        gradient = -2 * X.T.dot(y - X.dot(beta))
```

```
        # Update direction
        direction = update_direction(gradient, lambda_)

        beta = (1-alpha) * beta + alpha * direction

        # print(beta)
        # Calculate the objective value and append to the list
        objective_value = np.linalg.norm(y - X.dot(beta))**2
        objective_values.append(objective_value)

        # Append the current iterate (beta) to the list
        iterates.append(beta)

    return beta, iterates, objective_values


## Now let's test Frank-Wolfe

# initialization
beta_init = np.zeros(n)
t = 6
stepsize = 0.005
N_iter = 500

beta_fw, beta_fw_seq, fun_fw_seq = frank_wolfe(beta_init, X, y, stepsize, t, N_iter)

print(f"L1-constrained problem -- Error of Frank-Wolfe: \n{l2_error(beta_fw, beta_star)}

if l2_error(beta_fw, beta_star) < 0.05:
  print("\nFrank-Wolfe recovers the parameter acurrately \n")
else:
  print("\nThe error seems a bit large, perhaps you need to check the code or tune the h

# generate plots
plot_results(beta_fw_seq, fun_fw_seq, beta_star)
```

```
L1-constrained problem -- Error of Frank-Wolfe:
0.029385438828408498


Frank-Wolfe recovers the parameter acurrately
```



In [18]:
```
np.set_printoptions(precision=5, suppress=True)

print("beta[:20] for true beta: \n", beta_star[:20], "\n")
print("beta[:20] for L1-regularized problem from CVXPY: \n", beta_cvx_reg[:20], "\n")
```

```
print("beta[:20] for L1-constrained problem from CVXPY: \n", beta_cvx_con[:20], "\n")
print("beta[:20] for L1-regularized problem from Proximal Gradient: \n", beta_prox[:20],
print("beta[:20] for L0-constrained problem from Projected Gradient: \n", beta_proj[:20]
print("beta[:20] for L1-constrained problem from Frank-Wolfe: \n", beta_fw[:20], "\n")
```

```
beta[:20] for true beta:
 [1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

beta[:20] for L1-regularized problem from CVXPY:
 [ 0.99062  0.98741  0.9843   0.98057  0.98904 -0.      -0.       0.
  0.       0.       0.       0.      -0.       0.       0.      -0.
  0.       0.      -0.      -0.     ]

beta[:20] for L1-constrained problem from CVXPY:
 [ 0.99468  0.99164  0.99016  0.98548  0.99459 -0.00386 -0.       0.
  0.       0.00189  0.       0.      -0.       0.       0.      -0.
  0.       0.      -0.      -0.     ]

beta[:20] for L1-regularized problem from Proximal Gradient:
 [ 0.99565  0.99291  0.992    0.987    0.99625 -0.00468 -0.       0.
  0.       0.00328  0.       0.      -0.       0.00041  0.      -0.
  0.       0.      -0.      -0.     ]

beta[:20] for L0-constrained problem from Projected Gradient:
 [1.00453 1.00019 1.0029  0.99327 1.00403 0.      0.      0.      0.
 0.      0.      0.      0.      0.      0.      0.      0.      0.
 0.      0.     ]

beta[:20] for L1-constrained problem from Frank-Wolfe:
 [ 1.00319  0.9904   1.0077   0.99571  0.99708 -0.      -0.       0.
 -0.       0.       0.       0.      -0.       0.      -0.      -0.
  0.       0.      -0.      -0.     ]
```