Advanced Topics in Convex Optimization

Prof. Athanasios Liavas

Michail Theologitis

School of Electrical and Computer Engineering
Technical University of Crete
Greece

1. Theoretical Exercises

1. Let $\boldsymbol{c} \in \mathbb{R}^n$ and $\Delta_n = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \geq \mathbf{0}, \sum_{i=1}^n x_i = 1\}$. Solve the KKT to solve the problem:

$$\min_{x \in \Delta_n} \ \boldsymbol{c}^T \boldsymbol{x} \tag{1.1}$$

Solution. Problem equation (1.1) is a convex optimization problem since both the objective and constraint functions are convex. We can rewrite it as follows:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^T \boldsymbol{x} \\ \text{subject to} \quad & -\boldsymbol{x} \leq 0 \ , \ \mathbf{1}^T \boldsymbol{x} - 1 = 0 \end{aligned}$$

Take $\frac{1}{n}\mathbf{1} \in \mathbb{X}$, which means that Slater's condition is satisfied and strong duality holds.

$$\begin{aligned} \text{Lagrangian}: \quad & L(\boldsymbol{x}, \boldsymbol{\lambda}, v) = \boldsymbol{c}^T \boldsymbol{x} - \boldsymbol{\lambda}^T \boldsymbol{x} + v(\mathbf{1}^T \boldsymbol{x} - 1) \\ & \nabla L(\boldsymbol{x}, \boldsymbol{\lambda}, v) = \boldsymbol{c} - \boldsymbol{\lambda} + v\mathbf{1} \end{aligned}$$

Let $\boldsymbol{x}^*$ and $(\boldsymbol{\lambda}^*, v^*)$ be any primal and dual optimal points, respectively. Since strong duality holds, $\boldsymbol{x}^*$ minimizes $L(\boldsymbol{x}, \boldsymbol{\lambda}^*, v^*)$ and we have complementary slackness. The KKT conditions are the following:

$$\text{Stationarity}: \quad \nabla L(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, v^*) = 0 \ \Rightarrow \ c_i - \lambda_i^* + v^* = 0 \ \ \forall \ i \in [n] \tag{1.2}$$

$$\text{Slackness}: \quad \lambda_i^* v^* = 0 \ \ \forall \ i \in [n] \tag{1.3}$$

$$\text{Primal Feasibility}: \quad x_i^* \geq 0 \ \ \forall \ i \in [n] \tag{1.4}$$

$$\text{Primal Feasibility}: \quad \sum_{i=1}^n x_i^* = 1 \tag{1.5}$$

$$\text{Dual Feasibility}: \quad \lambda_i \geq 0 \ \ \forall \ i \in [n] \tag{1.6}$$

We also define:

$$\mathcal{I}_+(\boldsymbol{x}) = \{i \in [n] : x_i > 0\} \tag{1.7}$$

If $x_i^* > 0$ for some $i \in [n]$ then from equation (1.3) we have $\lambda_i = 0$ and from equation (1.2) we get $-v^* = c_i$.

$$-v^* = c_i \ \ \forall \ i \in \mathcal{I}_+(\boldsymbol{x}^*) \tag{1.8}$$

Combining equation (1.2) and equation (1.6) we get that $c_j + v^* \geq 0 \ \ \forall \ j \in [n]$, or, equivalently:

$$-v^* \leq \min_{j \in [n]} c_j \tag{1.9}$$

We substitute on equation (1.9) the result from equation (1.8) and get $c_i \leq \min_{j\in[n]} c_j \ \forall\ i \in \mathcal{I}_+(\boldsymbol{x}^*)$. Hence,

$$c_i = \min_{j\in[n]} c_j \ \ \forall\ i \in \mathcal{I}_+(\boldsymbol{x}^*) \tag{1.10}$$

Finally, we conclude that if $c_i > c_j$ for some $i, j \in [n]$ then it must be that $x_i^* = 0$. This follows directly from equation (1.10) by contradiction on the assumption $i \in \mathcal{I}_+(\boldsymbol{x}^*)$.

$$c_i > c_j,\ i, j \in [n] \ \Rightarrow\ x_i^* = 0 \tag{1.11}$$

Essentially, the result of solving the KKT for this optimization problem is captured succinctly in equation (1.11). Let's define:

$$\mathcal{I}_M = \{i \in [n] : c_i = c_{\min}\} \quad \text{with} \quad c_{\min} = \min_{j\in[n]} c_j \quad , \quad \mathcal{I}_M^C = [n] \setminus \mathcal{I}_M \tag{1.12}$$

Then, from equation (1.11) and with the help of the definitions in equation (1.12), we can describe the full family of minimizers of problem equation (1.1):

$$x_i^* = 0 \ \ \forall\ i \in \mathcal{I}_M^C \quad \text{and} \quad x_i^* > 0 \ \ \forall\ i \in \mathcal{I}_M \quad \text{with} \quad \sum_{i\in\mathcal{I}_M} x_i = 1 \tag{1.13}$$

Lastly, the optimum p^* is:

$$p^* = \boldsymbol{c}^T\boldsymbol{x} = \sum_{i\in\mathcal{I}_M^C} c_i x_i \ (\to 0) + \sum_{i\in\mathcal{I}_M} c_i x_i \overset{(1.12)}{=} c_{\min} \sum_{i\in\mathcal{I}_M} x_i \overset{(1.13)}{=} c_{\min}$$

---

2. Experimental-Algorithmic Exercises

2.1 Linear Programs

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ (with $m \ll n$), $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^n$. Consider the program in standard form:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \; f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x}, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq 0 \tag{2.1}$$

Generate a feasible problem, solve it via CVX, and compute $\boldsymbol{x}^*$ and f_{opt}. Express the problem in an equivalent form as follows:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \; f(\boldsymbol{x}) + g(\boldsymbol{z}), \quad \text{subject to} \quad \boldsymbol{x} = \boldsymbol{z}$$

where

$$g(\boldsymbol{z}) = \delta_{\mathbb{R}^n_+}(\boldsymbol{z})$$

and

$$\text{dom}(f) = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$$

Solve the problem via the ADMM algorithm, as follows:

1. Write down the ADMM updates.
2. Solve the optimization subproblems.
3. Implement the solutions efficiently.
4. Plot in a semilogy diagram quantities $f(\boldsymbol{x}^{(k)}) - f_{\text{opt}}$.

Solution. 1-3. We work as follows:

$$\begin{aligned}
&\min_{\boldsymbol{x}\in\mathbb{R}^n} \; \boldsymbol{c}^T\boldsymbol{x}, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b},\ \boldsymbol{x} \geq 0 \\
\Leftrightarrow \quad &\min_{\boldsymbol{x}\in\mathbb{R}^n} \; \boldsymbol{c}^T\boldsymbol{x} + \delta_{\{\boldsymbol{x}\in\mathbb{R}^n:\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}\}}(\boldsymbol{x}), \quad \text{subject to} \quad \boldsymbol{x} \geq 0 \\
\Leftrightarrow \quad &\min_{\boldsymbol{x}\in\mathbb{R}^n} \; \underbrace{\boldsymbol{c}^T\boldsymbol{x} + \delta_{\{\boldsymbol{x}\in\mathbb{R}^n:\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}\}}(\boldsymbol{x})}_{f(\boldsymbol{x})} + \underbrace{\delta_{\mathbb{R}^n_+}(\boldsymbol{x})}_{g(\boldsymbol{x})} \\
\Leftrightarrow \quad &\min_{\boldsymbol{x}\in\mathbb{R}^n} \; f(\boldsymbol{x}) + g(\boldsymbol{z}), \quad \text{subject to} \quad \boldsymbol{x} = \boldsymbol{z} \\
\Leftrightarrow \quad &\min_{\boldsymbol{x}\in\mathbb{R}^n} \; f(\boldsymbol{x}) + g(\boldsymbol{z}), \quad \text{subject to} \quad \boldsymbol{x} - \boldsymbol{z} = \boldsymbol{0}
\end{aligned} \tag{2.2}$$

From the problem formulation in [Bec17, Equation (15.1)] and (2.2) we have the following:

$$\begin{aligned}
&h_1(\boldsymbol{x}) = f(\boldsymbol{x}) = \boldsymbol{c}^T\boldsymbol{x} + \delta_{\{\boldsymbol{x}\in\mathbb{R}^n:\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}\}}(\boldsymbol{x}) \\
&h_2(\boldsymbol{z}) = g(\boldsymbol{z}) = \delta_{\mathbb{R}^n_+}(\boldsymbol{z}) \\
&\boldsymbol{A}_{\text{admm}} = \boldsymbol{I} \\
&\boldsymbol{B}_{\text{admm}} = -\boldsymbol{I} \\
&\boldsymbol{c}_{\text{admm}} = \boldsymbol{0}
\end{aligned}$$

Now, by replacing all above in the $\boldsymbol{x}^{(k+1)}$ update in [Bec17, ADMM 15.2] we have:

$$
\begin{aligned}
\boldsymbol{x}^{(k+1)} &\in \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \boldsymbol{c}^T\boldsymbol{x} + \delta_{\{\boldsymbol{x}\in\mathbb{R}^n:\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}\}}(\boldsymbol{x}) + \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{z}^{(k)} + \frac{1}{\rho}\boldsymbol{y}^{(k)}\|_2^2 \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \boldsymbol{c}^T\boldsymbol{x} + \frac{\rho}{2}\|\boldsymbol{x} \underbrace{-\boldsymbol{z}^{(k)} + \frac{1}{\rho}\boldsymbol{y}^{(k)}}_{-\boldsymbol{q}}\|_2^2, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \boldsymbol{c}^T\boldsymbol{x} + \frac{\rho}{2}\|\boldsymbol{x} - \boldsymbol{q}\|_2^2, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \langle \boldsymbol{c}\,,\,\boldsymbol{x}\rangle + \frac{\rho}{2}\|\boldsymbol{x}\|_2^2 - \rho\langle \boldsymbol{q}\,,\,\boldsymbol{x}\rangle + \cancel{\frac{\rho}{2}\|\boldsymbol{q}\|_2^2}^{\text{const.}}, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \frac{\rho}{2}\|\boldsymbol{x}\|_2^2 + \langle \boldsymbol{c} - \rho\boldsymbol{q}\,,\,\boldsymbol{x}\rangle, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \rho(\frac{1}{2}\|\boldsymbol{x}\|_2^2 + \langle \underbrace{\frac{1}{\rho}\boldsymbol{c} - \boldsymbol{q}}_{-\tilde{\boldsymbol{q}}}\,,\,\boldsymbol{x}\rangle), \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \frac{1}{2}\|\boldsymbol{x}\|_2^2 - \langle \tilde{\boldsymbol{q}}\,,\,\boldsymbol{x}\rangle, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \frac{1}{2}\|\boldsymbol{x}\|_2^2 - \langle \tilde{\boldsymbol{q}}\,,\,\boldsymbol{x}\rangle + \frac{1}{2}\|\tilde{\boldsymbol{q}}\|_2^2, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
&= \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmin}}\ \frac{1}{2}\|\boldsymbol{x} - \tilde{\boldsymbol{q}}\|_2^2, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\
&= P_{\{\boldsymbol{x}\in\mathbb{R}^n:\boldsymbol{A}\boldsymbol{x}=\boldsymbol{b}\}}(\tilde{\boldsymbol{q}}) \qquad \text{(From Optimization 1, Set 3, Ex. 5)} \\
&= \tilde{\boldsymbol{q}} - \boldsymbol{A}^\dagger(\boldsymbol{A}\tilde{\boldsymbol{q}} - \boldsymbol{b}) \quad , \quad \text{where} \quad \tilde{\boldsymbol{q}} = \boldsymbol{q} - \frac{1}{\rho}\boldsymbol{c} = \boldsymbol{z}^{(k)} - \frac{1}{\rho}(\boldsymbol{y}^{(k)} + \boldsymbol{c})
\end{aligned} \tag{2.3}
$$

Again, we follow the same procedure in the $\boldsymbol{z}^{(k+1)}$ update in [Bec17, ADMM 15.2] and we have:

$$
\begin{aligned}
\boldsymbol{z}^{(k+1)} &\in \underset{\boldsymbol{z}\in\mathbb{R}^n}{\operatorname{argmin}}\ \delta_{\mathbb{R}^n_+}(\boldsymbol{z}) + \frac{\rho}{2}\|\boldsymbol{z} - \boldsymbol{x}^{(k+1)} - \frac{1}{\rho}\boldsymbol{y}^{(k)}\|_2^2 \\
&= \underset{\boldsymbol{z}\in\mathbb{R}^n}{\operatorname{argmin}}\ \frac{\rho}{2}\|\boldsymbol{x}^{(k+1)} - \boldsymbol{z} + \frac{1}{\rho}\boldsymbol{y}^{(k)}\|_2^2 \quad \text{subject to} \quad \boldsymbol{z} \in \mathbb{R}^n_+ \\
&= \underset{\boldsymbol{z}\in\mathbb{R}^n}{\operatorname{argmin}}\ \frac{\rho}{2}\|\boldsymbol{z} \underbrace{-\boldsymbol{x}^{(k+1)} - \frac{1}{\rho}\boldsymbol{y}^{(k)}}_{-\boldsymbol{\ell}}\|_2^2 \quad \text{subject to} \quad \boldsymbol{z} \in \mathbb{R}^n_+ \\
&= \underset{\boldsymbol{z}\in\mathbb{R}^n}{\operatorname{argmin}}\ \frac{\rho}{2}\|\boldsymbol{z} - \boldsymbol{\ell}\|_2^2 \quad \text{subject to} \quad \boldsymbol{z} \in \mathbb{R}^n_+ \\
&= \operatorname{prox}_{\delta_{\mathbb{R}^n_+}}(\boldsymbol{\ell}) = P_{\delta_{\mathbb{R}^n_+}}(\boldsymbol{\ell}) \qquad \text{(From [BV04, p. 399])} \\
&= \max(\boldsymbol{\ell}, \boldsymbol{0}) \qquad \text{(Elementwise)} \quad , \quad \text{where} \quad \boldsymbol{\ell} = \boldsymbol{x}^{(k+1)} + \frac{1}{\rho}\boldsymbol{y}^{(k)}
\end{aligned} \tag{2.4}
$$

All in all, from (2.3), (2.4) and [Bec17, ADMM 15.2] we have the following ADMM updates:

1. $\boldsymbol{x}^{(k+1)}$ update:

$$
\begin{aligned}
\tilde{\boldsymbol{q}} &= \boldsymbol{z}^{(k)} - \frac{1}{\rho}(\boldsymbol{y}^{(k)} + \boldsymbol{c}) \\
\boldsymbol{x}^{(k+1)} &= \tilde{\boldsymbol{q}} - \boldsymbol{A}^\dagger(\boldsymbol{A}\tilde{\boldsymbol{q}} - \boldsymbol{b})
\end{aligned}
$$

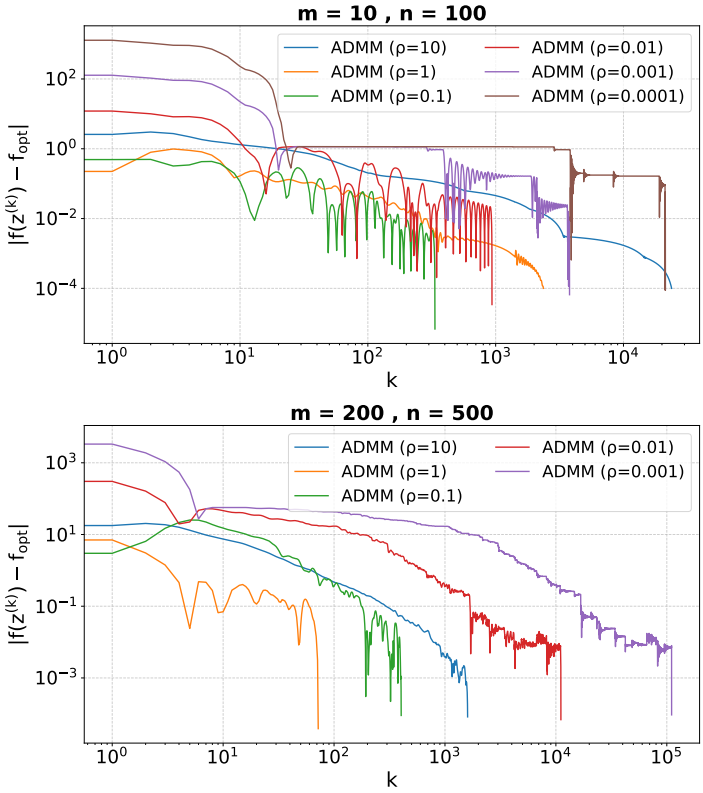

Figure 2.1: ADMM convergence path with varying ρ. Both x- and y-axis are logarithmic.

2. $\boldsymbol{z}^{(k+1)}$ update:

$$\boldsymbol{\ell} = \boldsymbol{x}^{(k+1)} + \frac{1}{\rho}\boldsymbol{y}^{(k)}$$
$$z_i^{(k+1)} = \max(\ell_i, 0)$$

3. $\boldsymbol{y}^{(k+1)}$ update:

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} + \rho(\boldsymbol{x}^{(k+1)} - \boldsymbol{z}^{(k+1)})$$

4. We construct $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{m\times n})$ and $\boldsymbol{c} \sim \mathcal{U}_{[0,1]}$. We sample $\boldsymbol{c}$ as nonnegative because we need to ensure that the problem is not unbounded (allowing negative values could create a scenario that allows $\boldsymbol{c}^T\boldsymbol{x} \to -\infty$ along some feasible directions). Finally, we construct a solution $\boldsymbol{x}_{\text{sol}} \sim \mathcal{U}_{[0,1]}$ and construct $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_{\text{sol}}$.

Notably, ADMM does not enforce all constraints immediately at each step, which means that $\boldsymbol{x}^{(k)}$ might be infeasible—at least, for the moment. It gradually enforces feasibility via $\boldsymbol{x} = \boldsymbol{z}$ and dual updates. Thus, early on, we avoid comparing $f(\boldsymbol{x}^{(k)})$ with f_{opt}. Instead we choose to compare $f(\boldsymbol{z}^{(k)})$ with f_{opt}. Finally, our stopping criterion becomes:

$$|f(\boldsymbol{z}^{(k)}) - f_{\text{opt}}| \leq 0.0001$$

We experiment with different values of the penalty parameter ρ and present the results in Figure 2.1. For the problem size $(n, m) = (10, 100)$ we observe that the best-performing value of ρ is 0.1. In contrast, for the larger problem with $(n, m) = (200, 500)$ the most effective choice is 1. For these optimal ρ values, we further illustrate the ADMM convergence trajectories in Figure 2.2. We notice oscillations which are common in ADMM.

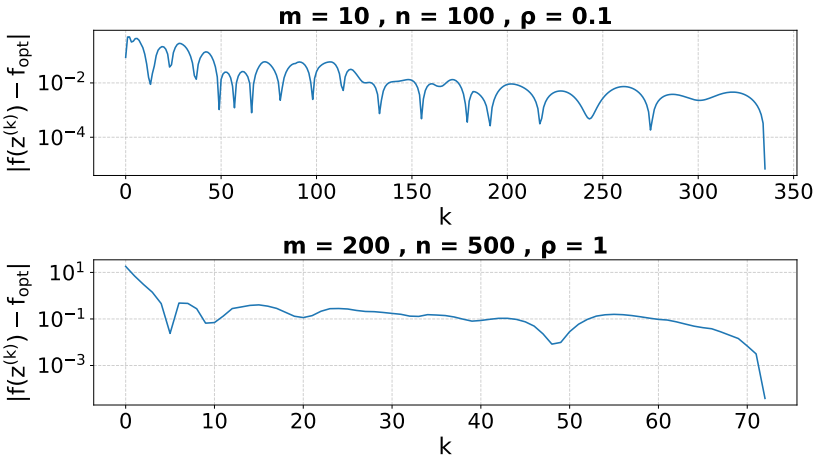

Figure 2.2: ADMM convergence path of best-performing ρ.

2.2 LASSO

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$. Generate random *sparse* vector $\boldsymbol{x}_s \in \mathbb{R}^n$ (let $s \ll n$ be the number of nonzero elements of $\boldsymbol{x}_s$ and let $m \gtrapprox 2s\log(n) \ll n$). Compute $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_s + \boldsymbol{e}$. Try to recover $\boldsymbol{x}_s$ as follows:

1. Use CVX, solve the problems (set $\lambda = 0.01, 0.1, 1, 10$)

$$\min_{x\in\mathbb{R}^n} \quad f_1(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_1 \tag{2.5}$$

$$\min_{x\in\mathbb{R}^n} \quad f_2(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2 + \lambda\|\boldsymbol{x}\|_2^2 \tag{2.6}$$

   and compute $\boldsymbol{x}^*$ and f_{opt}. Compare $\boldsymbol{x}^*$ and $\boldsymbol{x}_s$. What do you observe?

2. Solve problem equation (2.5) using the subgradient descent algorithm

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \frac{1}{\sqrt{k+1}\|\boldsymbol{g}^{(k)}\|_2}\boldsymbol{g}^{(k)} \quad , \quad \boldsymbol{g}^{(k)} \in \partial f_1(\boldsymbol{x}^{(k)}) \tag{2.7}$$

   Terminating condition: $f_1(\boldsymbol{x}^{(k)}) < cf_{\text{opt}}$, for $c \gtrapprox 1$. Plot (semilogy) quantity $f_1(\boldsymbol{x}^{(k)}) - f_{\text{opt}}$ versus k.

3. Solve the problem equation (2.5) using the algorithms ISTA and FISTA. Terminating condition: $f_1(\boldsymbol{x}^k) < c \cdot f_{\text{opt}}$ for $c \gtrapprox 1$. Plot, in a common semilogy, quantity $f_1(\boldsymbol{x}^k) - f_{\text{opt}}$ versus k for the three algorithms. What do you observe?

Solution. Essentially, in this exercise we investigate two different forms of regularization strategies in quadratic functions $f(x) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$. In problem equation (2.6), we see one of the simplest and most common kinds of parameter norm penalties known as weight decay, L^2 or Thikonov regularization. In this way, we drive the solution closer to the origin and show a preference for smaller components; the strength of this preference is proportional to λ. Notably, for $\lambda > 0$ the function f_2 is strongly convex ($\boldsymbol{A}^T\boldsymbol{A} + 2\lambda\boldsymbol{I} \succ 0$) and we have the following analytical solution (we require no assumptions on $\boldsymbol{A}$):

$$\boldsymbol{x}_2^* = \left(\boldsymbol{A}^T\boldsymbol{A} + 2\lambda\boldsymbol{I}\right)^{-1}\boldsymbol{A}^T\boldsymbol{b} \tag{2.8}$$

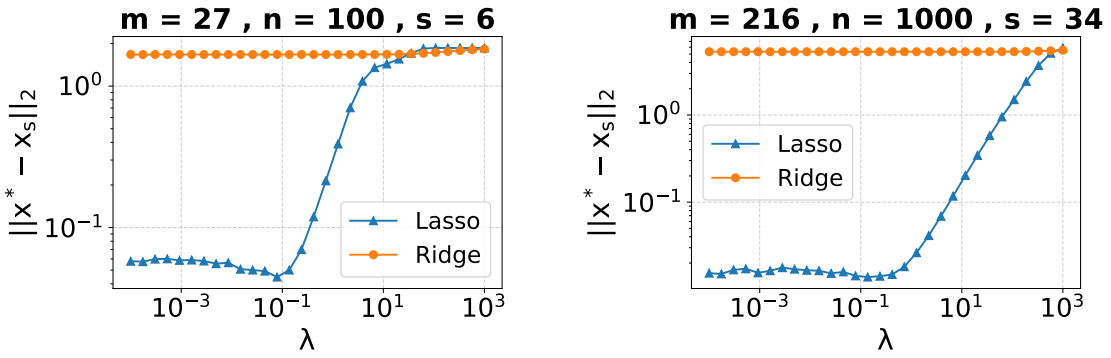

Figure 2.3: **Lasso** regression (L^1 regularization) vs. **Ridge** regression (L^2 regularization). Comparison between the recovery $\boldsymbol{x}^*$ of $\boldsymbol{x}_s$, for different values of λ, using the *euclidean distance*. Both axis are logarithmic.

In problem equation (2.5) we have L^1 regularization, which is also known as LASSO, which yields a sparse solution $\boldsymbol{x}_1^*$. In fact, by varying the parameter λ we can directly influence the sparsity of the solution; higher λ increases the penalty for any nonzero values in $\boldsymbol{x}_1^*$. Notably, the l_1 norm is commonly used as a convex surrogate for problems penalizing vector cardinality (l_0 norm)—the exact count of nonzero elements [BV04, pp. 308–310].

1. We construct $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{m\times n})$ and $\boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0}, 0.01 \cdot \boldsymbol{1})$. From the analytical solution equation (2.8) of problem equation (2.6) we notice that the result of L^2 regularization is the following: the addition of $2\lambda\boldsymbol{I}$ which essentially increases all eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ by 2λ. This shift in eigenvalues impacts the matrix inversion, affecting the solution $\boldsymbol{x}^*$. When the original eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$ are significantly lower than 2λ, the regularization term dominates. Conversely, if the original eigenvalues are much larger, varying λ within a typical range has a relatively minor effect on the solution.

We observe that L^1 regularization is preferred for the recovery of $\boldsymbol{x}_s$, as illustrated in Figure 2.3. This result is expected, given the sparse nature of $\boldsymbol{x}_s$, and the theoretical insights that ℓ_1-norm penalty promotes sparsity. Each different λ corresponds to a different cardinality of the solution $\boldsymbol{x}^*$, with values $\lambda \approx 10^{-1}$ performing best.

2. Using results from [Bec17, pp. 84–86] we first try to find some $\boldsymbol{g}^{(k)} \in \partial f_1(\boldsymbol{x}^{(k)})$.

$$\partial f_1(\boldsymbol{x}) = \partial\left(\frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2\right) + \partial\left(\lambda\|\boldsymbol{x}\|_1\right) = \left\{\boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\right)\right\} + \lambda\,\partial\left(\|\boldsymbol{x}\|_1\right) \tag{2.9}$$

Since $\operatorname{sgn}(\boldsymbol{x}) \in \partial\left(\|\boldsymbol{x}\|_1\right)$ it follows from equation (2.9) that

$$\boldsymbol{g}^{(k)} = \boldsymbol{A}^T\left(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}\right) + \lambda\operatorname{sgn}(\boldsymbol{x}^{(k)}) \;\in\; \partial f_1(\boldsymbol{x}^{(k)}) \tag{2.10}$$

We set $c = 1.1$. The results of subgradient descent for different values of λ are depicted in Figure 2.4. Notably, the values f_{opt} (and by extension the converged $f_1(\boldsymbol{x}^{(k)})$) have vastly different magnitudes for different λ values. We summarize this in Table 2.1.

3. We use the definition of the *soft thresholding* function from [Bec17, Example 6.8] with $\tau > 0$ which is the proximal operator of the ℓ_1-norm, i.e., $\operatorname{prox}_{\tau\|\cdot\|_1}(\boldsymbol{x}) = \mathcal{T}_\tau(\boldsymbol{x})$:

$$\mathcal{T}_\tau(\boldsymbol{x})_i = [|\boldsymbol{x}| - \tau\boldsymbol{e}]_+ \odot \operatorname{sgn}(\boldsymbol{x}) \tag{2.11}$$

Interestingly, many regularization strategies can be equivalently interpreted as MAP Bayesian inference with specific prior assumptions on $\boldsymbol{x}$. In particular, problem equation (2.6) (L^2 regularization) is equivalent to zero-mean Gaussian prior with covariance $\frac{1}{2\lambda}\boldsymbol{I}$, that is, $\mathbb{P}\left(\boldsymbol{x} \mid \boldsymbol{A}\right) = \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{0}, \frac{1}{2\lambda}\boldsymbol{I}\right)$. On the other hand, problem equation (2.5) (L^1 regularization) is equivalent to Laplacian prior for $\boldsymbol{x}$, that is, $\mathbb{P}\left(\boldsymbol{x} \mid \lambda\right) = \Pi_i\mathbb{P}\left(x_i \mid \lambda\right)$ with $\mathbb{P}\left(x_i \mid \lambda\right) = \text{Laplace}\left(x_i \mid 0, \frac{1}{2\lambda}\right)$ [Fig01], [GBC16, pp. 224–230]

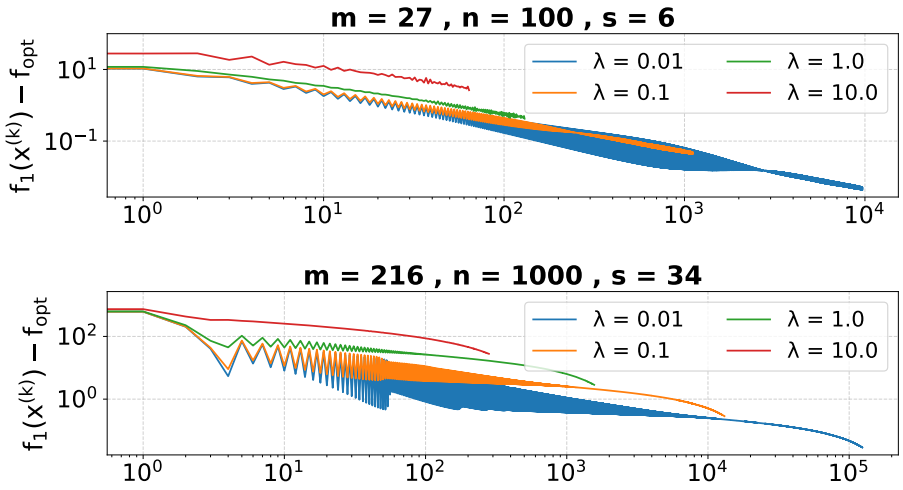

Figure 2.4: **Subgradient Descent**. Both axis are logarithmic

Table 2.1: The magnitudes of the optimal values for different values of λ, with $m = 27$, $n = 100$, and $s = 6$. Here, we observe that the magnitude $|f_1(\boldsymbol{x}^{(k)}) - f_{\text{opt}}|$ differs for different values of λ which helps explain the large differences in convergence magnitude observed in Figure 2.4

	$\lambda = 0.01$	$\lambda = 0.1$	$\lambda = 1$	$\lambda = 10$
f_{opt}	0.0441	0.4391	4.216	28.373
$f_1(\boldsymbol{x}^{(k)})$	0.0485	0.4830	4.638	31.008
$\lvert f_1(\boldsymbol{x}^{(k)}) - f_{\text{opt}}\rvert$	0.0044	0.0438	0.4219	2.6349

The ISTA algorithm has the following update rule [Bec17, p. 295]:

$$\boldsymbol{x}^{(k+1)} = \mathcal{T}_{\frac{\lambda}{L_k}}\left(\boldsymbol{x}^{(k)} - \frac{1}{L_k}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b})\right)$$

The FISTA algorithm has the following update rule ($\boldsymbol{y}^{(0)} = \boldsymbol{x}^{(0)}$, $t^{(0)} = 1$) [Bec17, p. 295]:

$$\begin{aligned}
\boldsymbol{x}^{(k+1)} &= \mathcal{T}_{\frac{\lambda}{L_k}}\left(\boldsymbol{y}^{(k)} - \frac{1}{L_k}\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{y}^{(k)} - \boldsymbol{b})\right) \\
t_{k+1} &= \frac{1 + \sqrt{1 + 4t_k^2}}{2} \\
\boldsymbol{y}^{(k+1)} &= \boldsymbol{x}^{(k+1)} + \left(\frac{t_k - 1}{t_{k+1}}\right)(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)})
\end{aligned}$$

We follow [Bec17, pp. 294–295] and set $L_k = \lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})$ for both algorithms. We also set $\lambda = 0.1$ which is the best-performing value for reconstructing $\boldsymbol{x}_s$ using CVX (see Figure 2.3). Additionally, we set $c = 1.01$.

The results of the three algorithms are shown in Figure 2.5. We notice that the optimization paths of ISTA and FISTA are smooth in comparison to our well-known, chaotic, and zigzag path of subgradient descent. However, subgradient descent is on par with ISTA on both sparsity examples ($s = 6$ and $s = 34$). Importantly, FISTA dominates in performance both ISTA and subgradient descent. It requires approximately $1000\times$ less iterations to converge. Lastly, a more nuanced observation is that ISTA and FISTA exhibit a sharper initial drop in the objective, compared to subgradient descent.

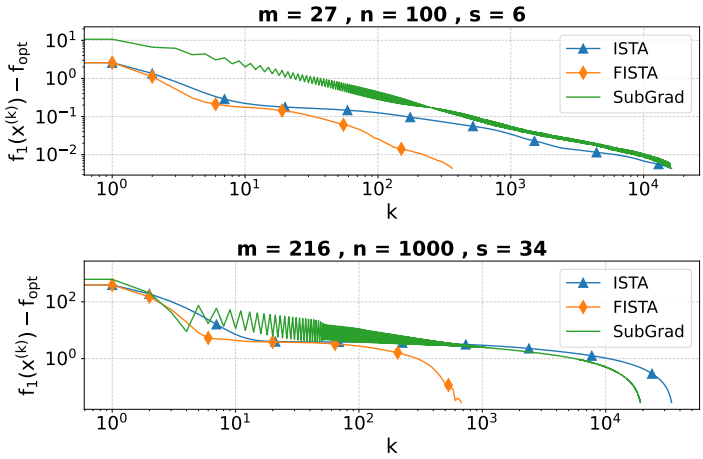

Figure 2.5: ISTA vs. FISTA vs. Subgradient Descent. We set $\lambda = 0.1$ and $c = 1.01$. Both axis are logarithmic

---

2.3 Basis Pursuit

Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$. Generate random *sparse* vector $\boldsymbol{x}_s \in \mathbb{R}^n$ (let $s \ll n$ be the number of nonzero elements of $\boldsymbol{x}_s$ and let $m \gtrapprox 2s\log(n) \ll n$). Compute $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_s$. Try to recover $\boldsymbol{x}_s$ by solving the problem

$$\min_{x \in \mathbb{R}^n} \; h(\boldsymbol{x}) = \|\boldsymbol{x}\|_1, \quad \text{subject to} \quad \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \tag{2.12}$$

1. Solve the problem using CVX and compute $\boldsymbol{x}^*$ and h_{opt}.
2. Solve the problem via the projected subgradient algorithm. Terminating condition: $h(\boldsymbol{x}^{(k)}) < ch_{\text{opt}}$, for $c \gtrapprox 1$. Plot (semilogy) quantity $h(\boldsymbol{x}^{(k)}) - h_{\text{opt}}$ versus k.
3. Compute a smooth approximation, h_μ, to h and solve the problem via the S-FISTA algorithm (see Example 10.59). Terminating condition: $h(\boldsymbol{x}^{(k)}) - h_{\text{opt}} < \epsilon$ (note that h is Lipschitz continuous, with constant $l_h = \sqrt{n}$.
4. Plot in the same semilogy quantity $h(\boldsymbol{x}^{(k)}) - h_{\text{opt}}$, for the two algorithms. What do you observe?

Solution. We construct $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{m \times n})$.

1. Done ✓. We consider two different sparsity settings: $(m, n, s) = (27, 100, 6)$ and $(m, n, s) = (214, 1000, 34)$

2. From [GBC16, Example 3.42] we know that $\text{sgn}(\boldsymbol{x}) \in \partial h(\boldsymbol{x})$. Also, from Problem 5 of the Exercise Set 3 in Optimization I, we proved that:

$$P_{\mathbb{S}}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{A}^\dagger(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \quad , \quad \text{where} \quad \mathbb{S} = \{\boldsymbol{x} \in \mathbb{R}^d : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\} \tag{2.13}$$

The projected subgradient descent has the following update rule [Bec17, p. 202, Theorem 8.28]:

$$\boldsymbol{x}^{(k+1)} = P_{\mathbb{S}}\left(\boldsymbol{x}^{(k)} - \frac{1}{\sqrt{k+1}\|\boldsymbol{g}^{(k)}\|_2}\boldsymbol{g}^{(k)}\right) \quad , \quad \boldsymbol{g}^{(k)} \in \partial f_1(\boldsymbol{x}^{(k)})$$

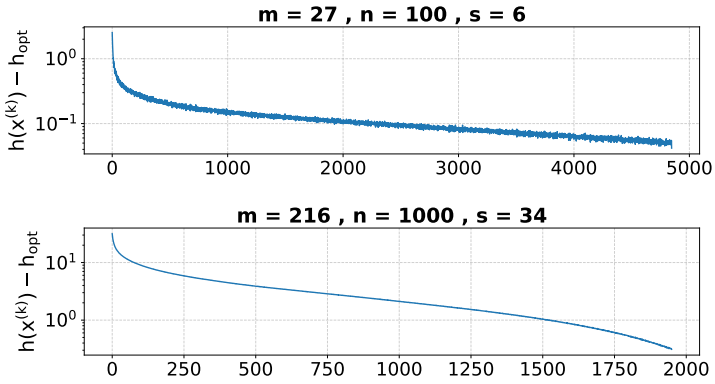

Figure 2.6: Projected Subgradient Descent. We set $c = 1.01$. The y-axis is logarithmic

Table 2.2: The projected subgradient descent convergence for different values of (m, n, s)

	$(27, 100, 6)$	$(214, 1000, 34)$
h_{opt}	4.4114	31.428
$h(\boldsymbol{x}^{(k)})$	4.453	31.739
$\lvert h(\boldsymbol{x}^{(k)}) - h_{\text{opt}}\rvert$	0.042	0.31
ϵ (for S-FISTA)	0.042	0.31

Firstly, notice that $\|\boldsymbol{g}^{(k)}\|_2 = \|\text{sgn}(\boldsymbol{x}^{(k)})\|_2 = \sqrt{n}$. Then, from Equation (2.13) the update rule becomes (we use an intermediary variable $\boldsymbol{u}$ so it is more compact):

$$\begin{aligned}\boldsymbol{u} &= \boldsymbol{x}^{(k)} - \frac{1}{\sqrt{n(k+1)}}\text{sgn}(\boldsymbol{x}^{(k)}) \\ \boldsymbol{x}^{(k+1)} &= \boldsymbol{u} - \boldsymbol{A}^{\dagger}(\boldsymbol{A}\boldsymbol{u} - \boldsymbol{b})\end{aligned}$$

In the algorithm we set $c = 1.01$. The results of the projected subgradient descent algorithm are shown in Figure 2.6. Notably, the choice of (m, n, s) affects the final magnitude of h_{opt} which in turn affects the final $|h(\boldsymbol{x}^{(k)}) - h_{\text{opt}}|$. We summarize this in Table 2.2.

3. We know the the Moreau Envelope M_h^{μ} is the $\frac{1}{\mu}$-smooth approximation of any Lipschitz convex function h. In our case, since the ℓ_1-norm is a convex Lipchitz function with constant $\ell_h = \sqrt{n}$, and following [Bec17, Example 10.54] we have that

$$h_{\mu}(\boldsymbol{x}) = M_h^{\mu}(\boldsymbol{x}) = \sum_{i=1}^{n} H_{\mu}(x_i)$$

is a $\frac{1}{\mu}$-smooth approximation of h with parameters $(1, \frac{n}{2})$, where H_{μ} is the Huber function defined—in the one dimensional case—as:

$$H_{\mu}(y) = \begin{cases} \frac{1}{2\mu}|y|^2, & |y| \leq \mu \\ |y| - \frac{\mu}{2}, & |y| > \mu \end{cases}$$

Additionally, following the analysis of [Bec17, Example 10.59] we can outline the S-FISTA algorithm. First, the choose $\boldsymbol{y}^{(0)} = \boldsymbol{x}^{(0)} \in \mathbb{S}$, $t_0 = 1$, and $\mu = \frac{\epsilon}{\ell_h^2}$. Then, we have the

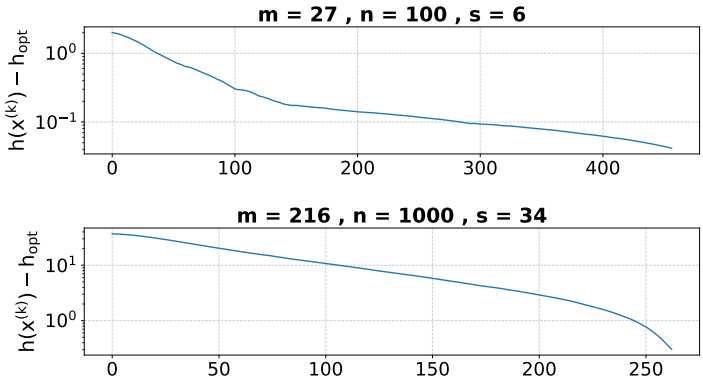

Figure 2.7: S-FISTA. We set $\epsilon = 0.042$ (top), $\epsilon = 0.31$ (bottom). The y-axis is logarithmic

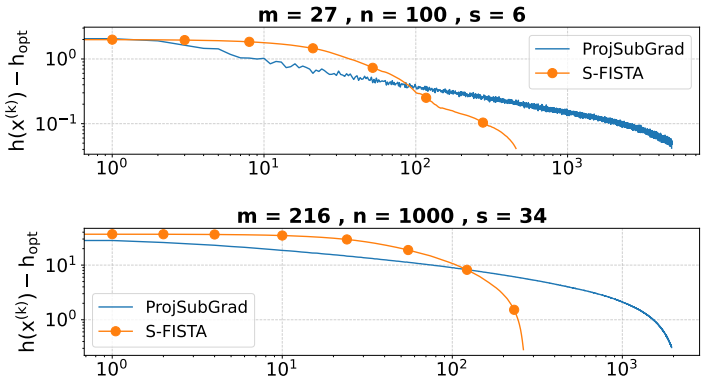

Figure 2.8: Projected Subgradient Descent vs. S-FISTA. We have the same stopping criterion. The y-axis is logarithmic

following update rule:

$$\begin{aligned}
\boldsymbol{u} &= \text{prox}_{\mu h}(\boldsymbol{y}^{(k)}) = \mathcal{T}_{\mu}(\boldsymbol{y}^{(k)}) \overset{(2.11)}{=} \dots \\
\boldsymbol{x}^{(k+1)} &= P_{\mathbb{S}}(\boldsymbol{u}) \overset{(2.13)}{=} = \boldsymbol{u} - \boldsymbol{A}^{\dagger}(\boldsymbol{A}\boldsymbol{u} - \boldsymbol{b}) \\
t_{k+1} &= \frac{1 + \sqrt{1 + 4t_k^2}}{2} \\
\boldsymbol{y}^{(k+1)} &= \boldsymbol{x}^{(k+1)} + \left(\frac{t_k - 1}{t_{k+1}}\right)(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)})
\end{aligned}$$

All that now remains is to choose an appropriate ϵ for the stopping criterion. However, since we want to compare S-FISTA with the projected subgradient descent we set ϵ as $|h(\boldsymbol{x}^{(k)}_{\text{subgrad}}) - h_{\text{opt}}|$ where $\boldsymbol{x}^{(k)}_{\text{subgrad}}$ is the convergence point for the projected subgradient descent with $c = 1.01$. We outline the two different ϵ in Table 2.2. The final results of S-FISTA are shown in Figure 2.7.

3. We plot the optimality gap $h(\boldsymbol{x}^{(k)}) - h_{\text{opt}}$ for both projected subgradient descent and S-FISTA in Figure 2.8. As expected from theory, S-FISTA converges significantly faster, requiring approximately $10\times$ fewer iterations. This is consistent with its convergence rate of $\mathcal{O}(1/k^2)$, compared to the slower $\mathcal{O}(1/\sqrt{k})$ of projected subgradient descent. We also observe that the subgradient method exhibits noticeable oscillations, whereas S-FISTA progresses smoothly.

2.4 Estimation of a piece-wise constant vector

Generate a piece-wise constant vector $\boldsymbol{x}_{\text{pwc}} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, and

$$\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_{\text{pwc}} + \boldsymbol{e}$$

Try to estimate $\boldsymbol{x}_{\text{pwc}}$ by solving, e.g., via CVX, the following problems (perform experiments with weak and strong noise):

1. the simple LS problem with $\min\limits_{x \in \mathbb{R}^n} f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2$;

2. the regularized LS problem

$$\min_{x \in \mathbb{R}^n} F(\boldsymbol{x}) = \underbrace{\frac{1}{2}\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2^2}_{f(\boldsymbol{x})} + \underbrace{\|\boldsymbol{D}\boldsymbol{x}\|_1}_{g(\boldsymbol{x})} \tag{2.14}$$

   where $\boldsymbol{D} \in \mathbb{R}^{(n-1) \times n}$ is given by

$$\boldsymbol{D} = \rho \begin{bmatrix} +1 & -1 & 0 & 0 & \dots & 0 \\ 0 & +1 & -1 & 0 & \dots & 0 \\ 0 & 0 & +1 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & +1 & -1 \end{bmatrix}$$

   with $\rho > 0$;

3. Solve problem (2.14) using the subgradient descent method. Terminating condition: $F(\boldsymbol{x}^{(k)}) < c \cdot F_{\text{opt}}$, for $c \gtrsim 1$. Plot (semilogy) quantity $F(\boldsymbol{x}^{(k)}) - F_{\text{opt}}$ versus k.

4. Solve problem (2.14) using the S-FISTA algorithm, and plot in a semilogy quantity $F(\boldsymbol{x}^k) - F_{\text{opt}}$ versus k. Terminating condition: STOP if $F(\boldsymbol{x}^{(k)}) - F_{\text{opt}} \leq \varepsilon$, for small positive ε. (see Example 10.60)

5. Put both plots in the same figure. What do you observe?

Solution.

1-2. We construct the matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with entries sampled independently from a standard normal distribution, i.e., $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{m \times n})$. We consider two levels of Gaussian noise: a weak noise setting with $\boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0}, 0.0848 \cdot \boldsymbol{I})$ and a strong noise setting with $\boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0}, 1.6389 \cdot \boldsymbol{I})$. The ground truth signal $\boldsymbol{x}_{\text{pcw}} \in \mathbb{R}^n$ is piecewise constant, consisting of 10 contiguous segments. Each segment corresponds to a block of consecutive entries in $\boldsymbol{x}_{\text{pcw}}$ that share the same constant value which is drawn from $\mathcal{N}(0, 1)$.

Here, we compare the quality of the solutions obtained from the regularized and the simple LS problems. In the regularized formulation, the parameter ρ plays a critical role in the solution, as it controls the strength of the total variation penalty. As illustrated in Figure 2.9, both excessively small and excessively large values of ρ lead to bad behavior: the solution becomes the same with the unregularized LS estimate. This observation remains consistent with both weak and strong noise. Lastly, we observe that the regularized LS method is either equivalent to or outperforms the simple LS solution. In particular, when ρ is appropriately chosen, the regularized formulation can achieve substantially better results–with 10×-50× lower error in the estimated solution.

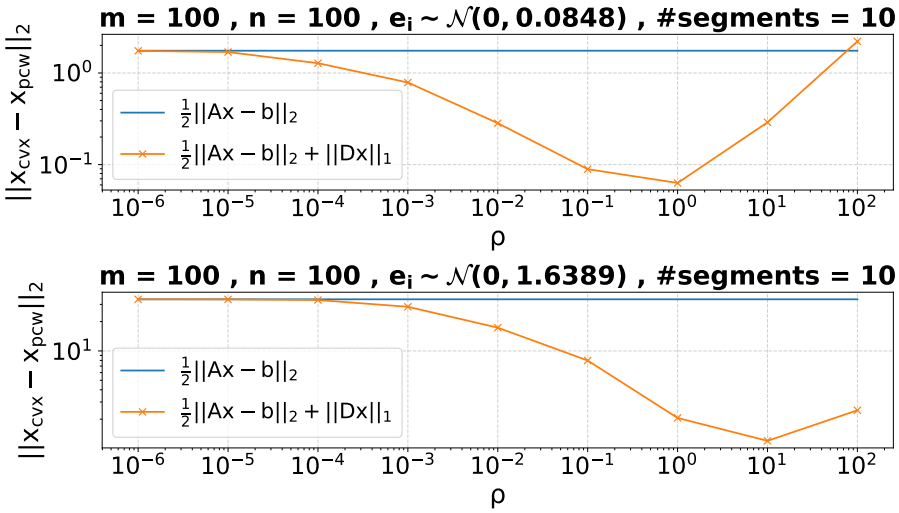

Figure 2.9: Regularized vs. Simple LS problem. Solution via CVX. We investigate different values of ρ for the regularized version and plot the results. We examine two noise levels: weak noise (top) with $\boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0}, 0.0848 \cdot \boldsymbol{I})$ and strong noise (bottom) with $\boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0}, 1.6389 \cdot \boldsymbol{I})$. The number of segments is the number of blocks of $\boldsymbol{x}_{\text{pcw}}$ that have the same value.

3. It is well-known that $\nabla f(\boldsymbol{x}) = \boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$ which also means

$$\partial f(\boldsymbol{x}) = \{\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\}$$

Furthermore, from [Bec17, Example 3.42] we know that $\text{sgn}(\boldsymbol{x}) \in \partial\|\boldsymbol{x}\|_1$ and from the weak affine transformation result of sub-differential calculus in [Bec17, Theorem 3.43] we have that

$$\boldsymbol{D}^T\text{sgn}(\boldsymbol{D}\boldsymbol{x}) \in \partial g(\boldsymbol{x})$$

Lastly, since $F(\boldsymbol{x}) = f(\boldsymbol{x}) + g(\boldsymbol{x})$, and using [Bec17, Theorem 3.36] we conclude:

$$\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \boldsymbol{D}^T\text{sgn}(\boldsymbol{D}\boldsymbol{x}) \in \partial F(\boldsymbol{x}) \tag{2.15}$$

Using (2.15) we can always compute a derivative of F given some point $\boldsymbol{x}$. All in all, the subgradient descent method with dynamic step size has the following update rule (see (2.7)):

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - \frac{1}{\sqrt{k+1}\|\boldsymbol{g}^{(k)}\|_2}\boldsymbol{g}^{(k)} \quad , \quad \boldsymbol{g}^{(k)} \in \partial F(\boldsymbol{x}^{(k)})$$

4. We use the results from [Bec17, Example 10.60] which is the same minimization with us, if we set $\lambda = 0$. Thus, we have:

$$\|Q\|_{2,2}^2 = \lambda_{\max}(\boldsymbol{Q}^T\boldsymbol{Q}) \quad \text{[Bec17, Example 1.1]}$$

$$\mu = \frac{2\|\boldsymbol{D}\|_{2,2}}{\sqrt{n-1}} \cdot \frac{\varepsilon}{\sqrt{\|\boldsymbol{D}\|_{2,2}^2(n-1)} + \sqrt{\|\boldsymbol{D}\|_{2,2}^2(n-1) + 2\|\boldsymbol{A}^T\boldsymbol{A}\|_{2,2}\varepsilon}}$$

$$\tilde{L} = \|\boldsymbol{A}\|_{2,2}^2 + \frac{\|\boldsymbol{D}\|_{2,2}^2}{\mu}$$

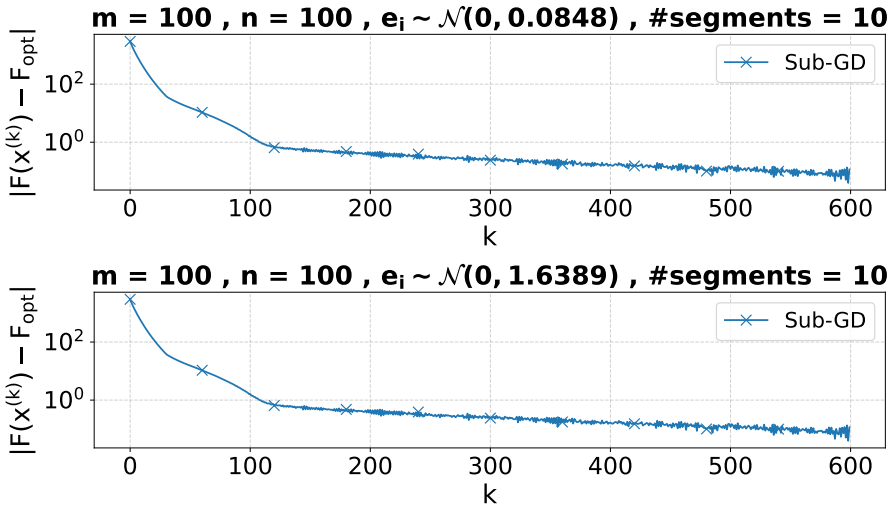

Figure 2.10: Subgradient Descent ($c = 1.02$). Weak noise (top) and strong noise (bottom)

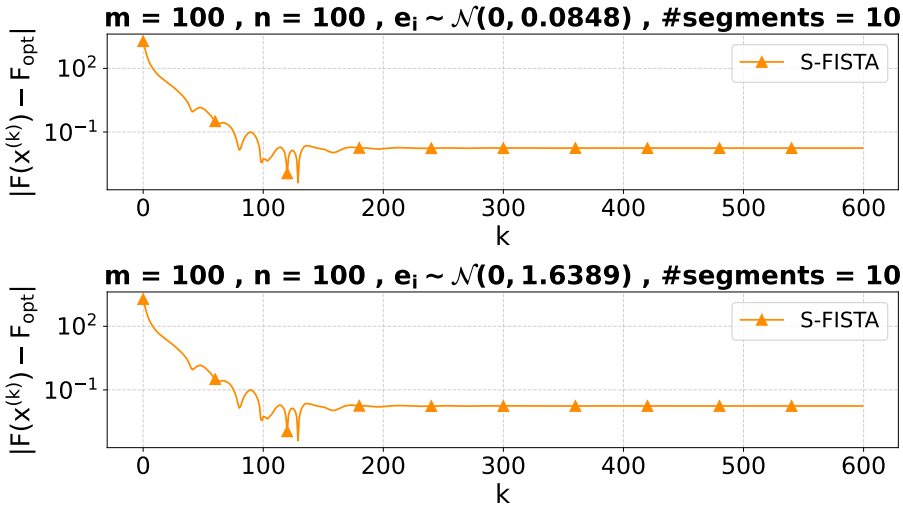

Figure 2.11: S-FISTA with final $\varepsilon = 0.02$. Weak noise (top) and strong noise (bottom)

Given the constants outlined above, we can now write the S-FISTA algorithm. First, the choose $\boldsymbol{y}^{(0)} = \boldsymbol{x}^{(0)} \in \mathbb{R}^n$. Then, we have the following update rule:

$$\begin{aligned}
\boldsymbol{x}^{(k+1)} &= \boldsymbol{y}^{(k)} - \frac{1}{\tilde{L}}\left(\nabla f(\boldsymbol{y}^{(k)}) + \frac{1}{\mu}\boldsymbol{D}^T\left(\boldsymbol{D}\boldsymbol{y}^{(k)} - \mathcal{T}_\mu(\boldsymbol{D}\boldsymbol{y}^{(k)})\right)\right) \\
t_{k+1} &= \frac{1+\sqrt{1+4t_k^2}}{2} \\
\boldsymbol{y}^{(k+1)} &= \boldsymbol{x}^{(k+1)} + \left(\frac{t_k - 1}{t_{k+1}}\right)(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)})
\end{aligned}$$

Of course, $\mathcal{T}_\mu$ is the soft-thresholding operator which has already been outlined in (2.11).

5. When comparing both methods in the same plot, we observe that S-FISTA performs considerably better than subgradient descent. The convergence of S-FISTA is significantly faster, with a smoother and more stable decrease. In contrast, subgradient descent exhibits slower convergence with pronounced oscillations, especially in later iterations.

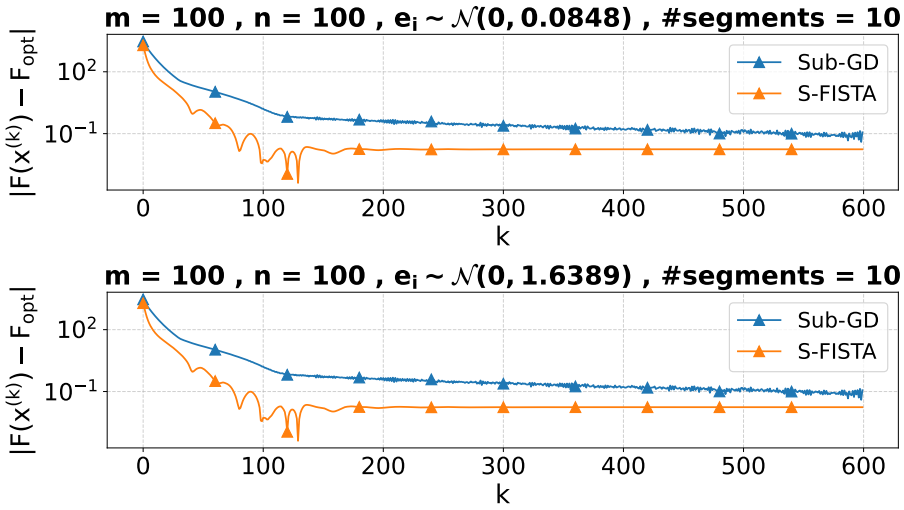

Figure 2.12: Subgradient Descent (SG-D) vs. S-FISTA with $\varepsilon = 5$

2.5 Matrix min l_1

Let $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ ($m \gg n$, $m \gtrapprox n$) and $\boldsymbol{b} \in \mathbb{R}^m$, and consider the problem:

$$\min_{x \in \mathbb{R}^n} \; f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1$$

Generate $\boldsymbol{A}$ and $\boldsymbol{b}$ using (1) Gaussian distribution, and (2) uniform distribution.

1. Solve the problem via CVX and compute $\boldsymbol{x}^*$ and f_{opt}.
2. Solve the problem with the subgradient descent algorithm with dynamic stepsize. As subgradient, use the "weak result"

   $$\boldsymbol{A}^T \text{sgn}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \in \partial f(\boldsymbol{x})$$

   Terminating condition $f(\boldsymbol{x}^{(k)}) \leq c \cdot f_{\text{opt}}$, for $c \gtrapprox 1$. Plot $f_{\text{best}}^{(k)} - f_{\text{opt}}$ versus k.
3. Implement the subgradient descent algorithm with Polyak step (see Example 3.44). Terminating condition $f(\boldsymbol{x}^{(k)}) \leq c \cdot f_{\text{opt}}$, for $c \gtrapprox 1$ (for example, $c = 1.1, 1.01, 1.001$). Plot $f_{\text{best}}^{(k)} - f_{\text{opt}}$ versus k.
4. Put the two plots in the same figure (experiment with $m \gg n$ and $m \gtrapprox n$ and Gaussian and Uniform distributions). What do you observe?
5. Compute L_f such that $\|\boldsymbol{g}\|_2 \leq L_f$ for all $\boldsymbol{g} \in \partial f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^n$ (see Assumption 8.12). Compute and plot the upper bound (see Theorem 8.13(c))

   $$f_{\text{best}}^{(k)} - f_{\text{opt}} \leq \frac{L_f \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\|_2}{\sqrt{k+1}}$$

   Plot the upper bound and the quantities ploted in question 4. in the same semilogy.
6. Solve the problem with the stochastic gradient algorithm. Compute the cost function every epoch (1 epoch is equal to m stochastic gradient iterations), i.e, $f(\boldsymbol{x}^{(km)})$. Run the algorithm for as many epochs as the number of iterations required for convergence in step (c) and put in semilogy quantity $f(\boldsymbol{x}^{(km)}) - f_{\text{opt}}$ versus k (k is the number of epochs).

7. Solve the problem with the incremental algorithm. Run the algorithm for as many outer iterations as the number of iterations required for convergence in step (c) and produce the corresponding semilogy.

8. Put all plots together.

Solution.

1-4. The subgradient update rule is [Bec17, Equation (8.3)]:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - t_k \boldsymbol{g}^{(k)}, \quad \text{where} \quad \boldsymbol{g}^{(k)} = \boldsymbol{A}^T \operatorname{sgn}(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}) \in \partial f(\boldsymbol{x})$$

Moreover, the classic dynamic stepsize rule is [Bec17, Theorem 8.28]:

$$t_k = \begin{cases} \frac{1}{\|\boldsymbol{g}^{(k)}\|_2 \sqrt{k+1}}, & \boldsymbol{g}^{(k)} \neq \boldsymbol{0} \\ \frac{1}{L_f}, & \boldsymbol{g}^{(k)} = \boldsymbol{0} \end{cases}$$

Conversely, Polyak's stepsize rule is [Bec17, Equation (8.13)]:

$$t_k = \begin{cases} \frac{f(\boldsymbol{x}^{(k)}) - f_{\text{opt}}}{\|\boldsymbol{g}^{(k)}\|_2^2}, & \boldsymbol{g}^{(k)} \neq \boldsymbol{0} \\ 1, & \boldsymbol{g}^{(k)} = \boldsymbol{0} \end{cases}$$

In Figure 2.13 we observe that increasing the number of rows m significantly improves convergence. As m grows relative to n, the problem becomes more overdetermined and better conditioned, which helps both the dynamic and Polyak stepsize algorithms converge faster. Conversely, when $m \approx n$, the problem is closer to being ill-posed or underdetermined, and convergence becomes slower and more erratic.

Interestingly, initializing $\boldsymbol{A}$ and $\boldsymbol{b}$ using a uniform distribution seems to result in slightly faster convergence compared to the Gaussian case, possibly due to more evenly bounded values that lead to more stable subgradients.

Furthermore, Polyak's stepsize clearly outperforms the dynamic rule, as shown in Figure 2.14. This makes sense: when we have access to the optimal value f_{opt}, Polyak's rule uses this information and makes much more informed step sizes to quickly zero in on the solution. This makes Polyak not just faster but also more stable. However, in practice, we do not have the f_{opt} information.

5. We are asked to compute L_f such that

$$\|\boldsymbol{g}\|_2 \leq L_f, \quad \forall \boldsymbol{g} \in \partial f(\boldsymbol{x})$$

From [Bec17, Theorem 3.61], the following statements are equivalent for a convex and proper function f:

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L_f \|\boldsymbol{x} - \boldsymbol{y}\|, \ \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \quad \Leftrightarrow \quad \|\boldsymbol{g}\|_* \leq L_f, \ \forall \boldsymbol{g} \in \partial f(\boldsymbol{x})$$

Since the dual norm of the ℓ_2-norm is again the ℓ_2-norm, bounding all subgradients in ℓ_2-norm is equivalent to showing that f is L_f-Lipschitz with respect to $\|\cdot\|_2$. Thus, to compute such an L_f, it suffices to find a constant satisfying:

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L_f \|\boldsymbol{x} - \boldsymbol{y}\|_2, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$$

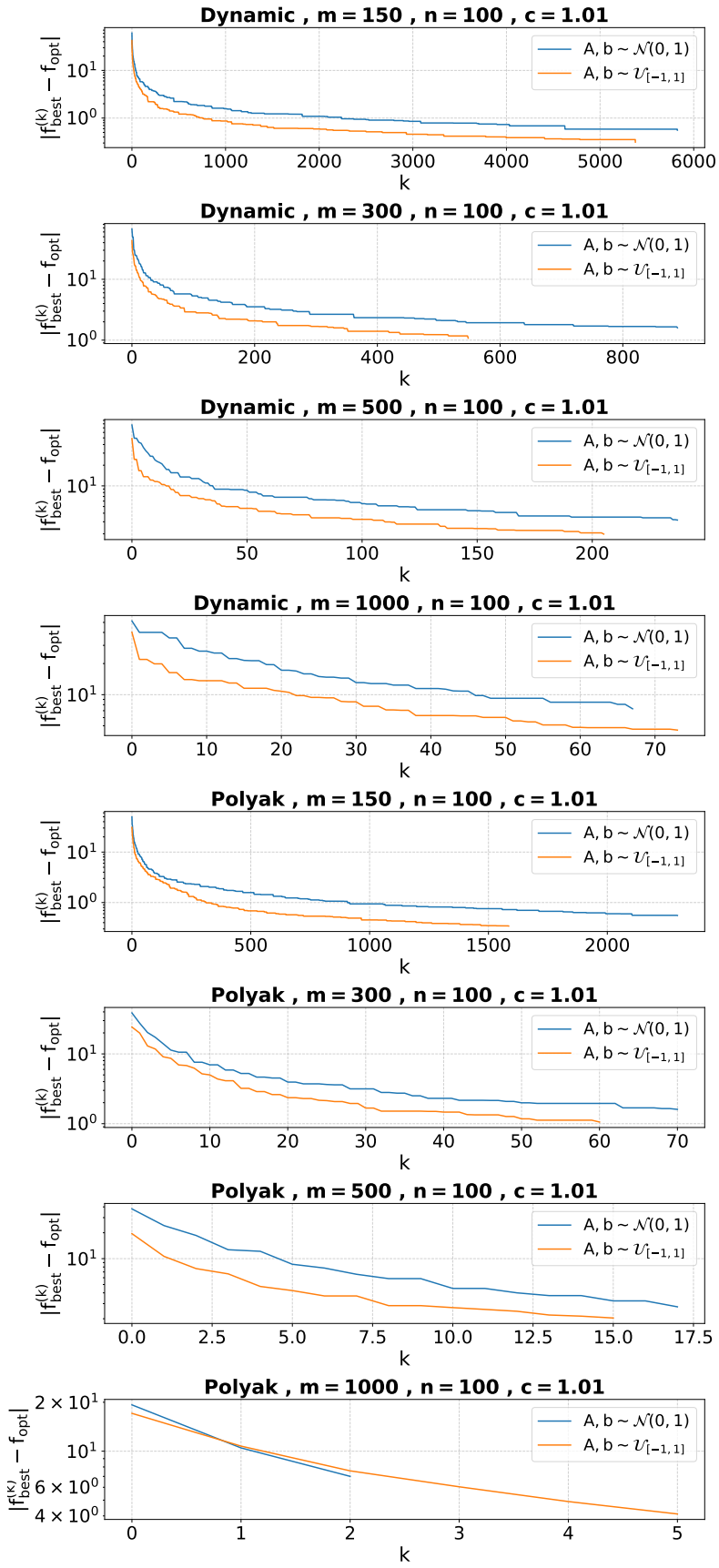

Figure 2.13: Subgradient Descent with Polyak stepsize (top four plots) and Dynamic Stepsize (bottom four plots). We investigate sampling $\boldsymbol{A}$ and $\boldsymbol{b}$ from either a gaussian or uniform distribution

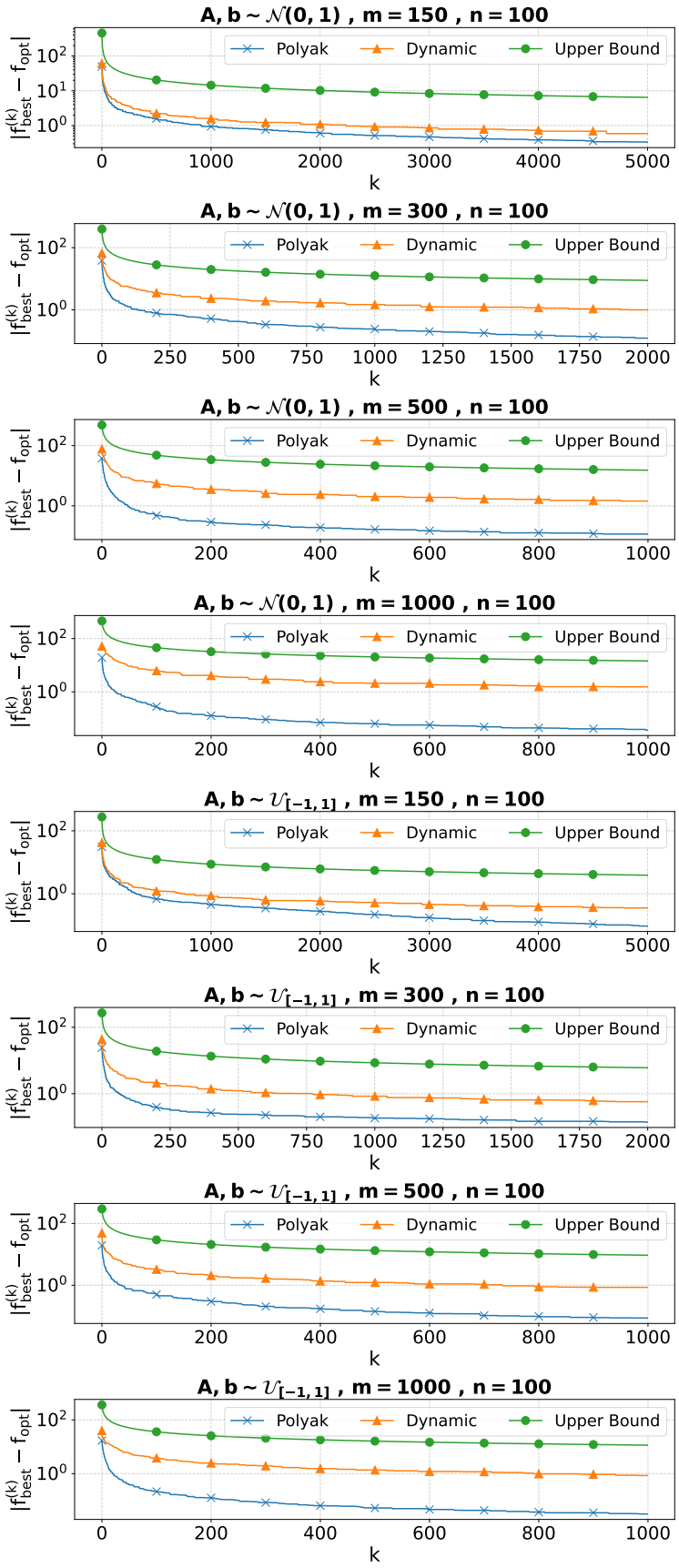

Figure 2.14: Subgradient Descent: Polyak vs. Dynamic stepsize. We also plot the Upper Bound given by the theoretical analysis and Equation (2.16). We investigate sampling $\boldsymbol{A}$ and $\boldsymbol{b}$ from either a gaussian (top four plots) or uniform (bottom four plots) distribution

We work as follows: let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, then:

$$\begin{aligned}
|f(\boldsymbol{x}) - f(\boldsymbol{y})| &= |\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1 - \|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}\|_1| \quad \text{(using the reverse triangle inequality)} \\
&\leq \|\boldsymbol{A}\boldsymbol{x} \cancel{- \boldsymbol{b}} - \boldsymbol{A}\boldsymbol{y} \cancel{+ \boldsymbol{b}}\|_1 \\
&= \|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\|_1 \qquad \text{(using Equation (3.1) with } q = 2 > p = 1) \\
&\leq m^{\frac{1}{1}-\frac{1}{2}} \|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\|_2 \\
&= \sqrt{m}\|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\|_2 \qquad \text{(From [Bec17, p. 7])} \\
&\leq \underbrace{\sqrt{m}\|\boldsymbol{A}\|_{2,2}}_{L_f} \|\boldsymbol{x} - \boldsymbol{y}\|_2
\end{aligned}$$

All in all,

$$L_f = \sqrt{m}\|\boldsymbol{A}\|_{2,2} \overset{\text{[Bec17, Example 1.1]}}{=} \sqrt{m \cdot \lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})}$$

$$UB^{(k)} = \frac{L_f \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\|_2}{\sqrt{k+1}} \qquad \text{(Upper Bound)} \tag{2.16}$$

In Figure 2.14, we plot the Upper Bound given by the equation above and confirm that it successfully bounds $f_{\text{best}}^{(k)} - f_{\text{opt}}$.

6-8 Let:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \vdots \\ \boldsymbol{a}_m^T \end{bmatrix} \quad , \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Following [Bec17, p. 226] we decompose function f as follows:

$$f(\boldsymbol{x}) = \sum_{i=1}^{m} f_i(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1, \quad \text{where} \quad f_i(\boldsymbol{x}) = |\boldsymbol{a}_i^T\boldsymbol{x} - b_i|$$

We can now write down the two algorithms. The *stochastic subgradient algorithm* [Bec17, p. 222] and the *incremental algorithm* [Bec17, p. 229] are outlined in Algorithm 1. Their only difference lies in how we choose the derivatives at each step: the stochastic algorithm chooses them randomly (highlighted in green), whereas the incremental chooses them in a cyclic manner (highlighted in red).

When implementing the two algorithms, we encountered a striking issue: they failed to converge and, in many cases, exhibited a prolonged upward trajectory in the objective function. This behavior led us to hypothesize that the individual subgradients (Line 7 of Algorithm 1)

$$\boldsymbol{g}_i^{(k,s)} = \boldsymbol{a}_i \operatorname{sgn}(\boldsymbol{a}_i^T\boldsymbol{x}^{(k,s)} - b_i) \in \partial f_i(\boldsymbol{x}^{(k,s)})$$

lack the necessary information to effectively minimize the global objective f. In other words, blindly following these subgradients does not consistently lead to meaningful descent steps.

To address this, we draw inspiration from the SGD literature [GBC16] and introduce the concept of mini-batching. Rather than immediately applying the subgradient in Line 9 of Algorithm 1, we collect B such subgradients into a batch and compute their average. This averaged subgradient is then used to perform the update step. The rationale is that averaging reduces the variance of the direction and better approximates the true descent direction for the full objective. Essentially, it utilizes more information.

Algorithm 1 Stochastic Subgradient & Incremental Algorithm [Bec17, pp. 225, 229]

Input: Initial $\boldsymbol{x}^{(0)} \in \mathbb{R}^n$

Compute $\tilde{L}_f = \sqrt{m \sum_{i=1}^{m} L_{f_i}^2}$, where $L_{f_i} = \|\boldsymbol{a}_i\|_2$ ▷ From [Bec17, p. 226]
for each epoch $k = 0, ..., E-1$ **do**
  Set $\boldsymbol{x}^{(k,0)} = \boldsymbol{x}^{(k)}$
  for each step $s = 0, ..., m-1$ **do**
    Sample index $i \in \{1, 2, ..., m\}$ randomly via uniform distribution
    Set $i = s + 1$
    Compute $\boldsymbol{g}_i^{(k,s)} \in \partial f_i(\boldsymbol{x}^{(k,s)})$
    Compute $t_{k,s} = (m\sqrt{2})/(\tilde{L}_f \sqrt{km + s + 1})$
    Update $\boldsymbol{x}^{(k,s+1)} = \boldsymbol{x}^{(k,s)} - t_{k,s}\, \boldsymbol{g}_i^{(k,s)}$
  Set $\boldsymbol{x}^{(k)} = \boldsymbol{x}^{(k,m)}$ ▷ $\boldsymbol{x}^{(k)}$ is our solution after $k+1$ epochs
  Compute current loss $f(\boldsymbol{x}^{(k)}) = \|\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}\|_1$
return $\boldsymbol{x}^{(E-1)}$

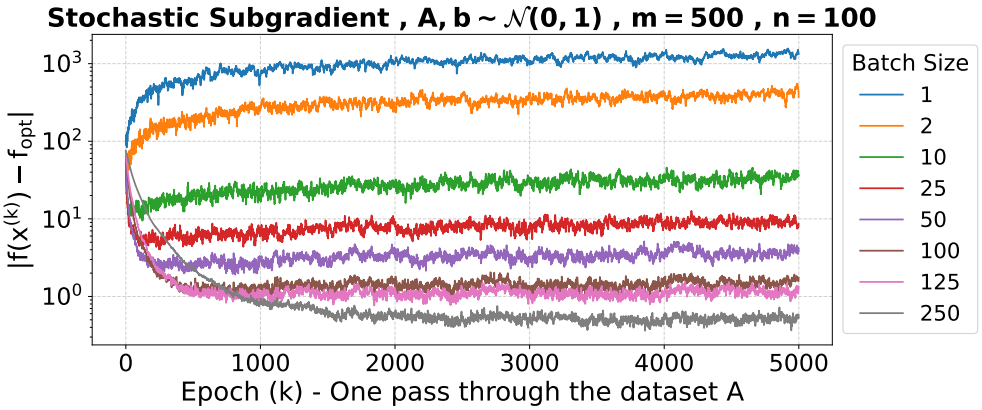

Figure 2.15: Investigation of Batch Size value for the Stochastic Subgradient algorithm

We explored this mini-batch approach using a variety of batch sizes for a fixed problem size of $(m, n) = (500, 100)$, running each configuration for an exhaustive number of epochs. Here, a single epoch corresponds to one complete pass through the dataset matrix $\boldsymbol{A}$. For instance, with a batch size of 50, an epoch consists of 10 updates (i.e., 10 batches of 50 samples), while with a batch size of 250, only 2 updates are performed per epoch.

The results of this investigation are illustrated in Figure 2.15. We observe that larger batch sizes generally yield better convergence behavior. In particular, convergence quality improves monotonically with increasing batch size. On the other hand, very small batch sizes (e.g., < 10) demonstrate poor and highly erratic performance. Based on these findings, we adopt a batch size of 25 for all subsequent experiments.

Lastly, in Figure 2.16, we present a comparison of all four algorithms considered in this study. Most notably, we observe that the incremental algorithm underperforms significantly compared to stochastic subgradient descent. This result is somewhat surprising: the cyclic and deterministic selection of subgradients appears to hurt convergence, whereas the inherent randomness in stochastic descent seems to help more effective minimization. Interestingly, the performance gap between the two is quite pronounced. Among all methods, subgradient descent with Polyak's stepsize achieves the best convergence. This is expected, as it leverages prior knowledge of the optimum f_{opt}, which is not available to the other algorithms and acts as an oracle on how much to "trust" current subgradients.

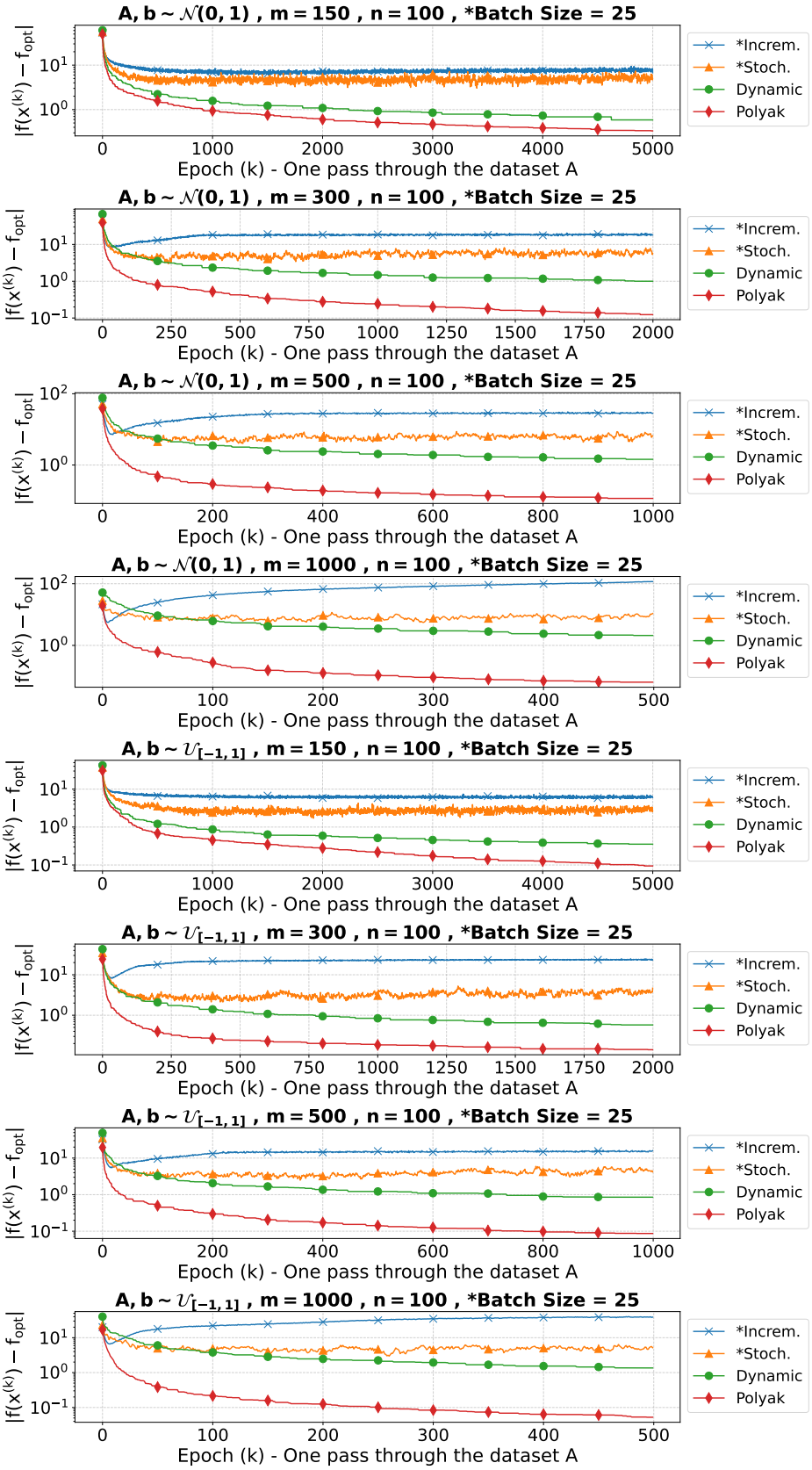

Figure 2.16: Stochastic Subgradient Descent (Stoch.) vs. Incremental Algorithm (Increm.) vs. Subgradient Descent with Polyak stepsize (Polyak) vs. Subgradient Descent with Dynamic stepsize (Dynamic). We use a batch size of 25 in *Stoch* and *Increm.*

2.6 l_1 regularized matrix min l_1

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ (with $m \ll n$), $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^n$. Consider the problem[1]:

$$\min_{x\in\mathbb{R}^n} \; H(\boldsymbol{x}) = h(\boldsymbol{x}) + \lambda g(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1 + \lambda\|\boldsymbol{x}\|_1$$

Solve the problem via CVX and compute $\boldsymbol{x}^*$ and H_{opt}. Solve the problem with (terminating condition for all algorithms: STOP if $H(\boldsymbol{x}^k) \leq c \cdot H_{\text{opt}}$, for $c \gtrapprox 1$. Test the cases $m \gg n$, $m \approx n$, and $m \ll n$.

1. The subgradient descent method (with both Polyak and dynamic step);
2. The proximal gradient method (see Example 9.29);
3. Smooth function $h(\boldsymbol{x})$ and solve the resulting problem with the S-FISTA algorithm (see example 10.60 for an "analogous" problem);
4. Smooth both functions and solve the resulting pronblem with the accelerated gradient algorithm (S-FISTA with no proximal step).

Solution.

1. From [GBC16, Example 3.42] we know that $\text{sgn}(\boldsymbol{x}) \in \partial g(\boldsymbol{x})$. Furthermore, $h(\boldsymbol{x}) = g(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$ and from the weak affine transformation result of sub-differential calculus in [GBC16, Theorem 3.43] we have that:

$$\boldsymbol{A}^T\partial g(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \subseteq \partial h(\boldsymbol{x}) \;\; \Rightarrow \;\; \boldsymbol{A}^T\text{sgn}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \subseteq \partial h(\boldsymbol{x})$$

All in all, we have that:

$$\boldsymbol{A}^T\text{sgn}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \lambda\text{sgn}(\boldsymbol{x}) \in \partial H(\boldsymbol{x})$$

The subgradient update rule becomes [Bec17, Equation (8.3)]:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - t_k\boldsymbol{g}^{(k)}, \quad \text{where} \quad \boldsymbol{g}^{(k)} = \boldsymbol{A}^T\text{sgn}(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b}) + \lambda\text{sgn}(\boldsymbol{x}^{(k)}) \in \partial H(\boldsymbol{x})$$

Moreover, the classic dynamic stepsize rule is [Bec17, Theorem 8.28]:

$$t_k = \begin{cases} \frac{1}{\|\boldsymbol{g}^{(k)}\|_2\sqrt{k+1}}, & \boldsymbol{g}^{(k)} \neq \boldsymbol{0} \\ \frac{1}{L_H}, & \boldsymbol{g}^{(k)} = \boldsymbol{0} \end{cases}$$

Conversely, Polyak's stepsize rule is [Bec17, Equation (8.13)]:

$$t_k = \begin{cases} \frac{H(\boldsymbol{x}^{(k)}) - H_{\text{opt}}}{\|\boldsymbol{g}^{(k)}\|_2^2}, & \boldsymbol{g}^{(k)} \neq \boldsymbol{0} \\ 1, & \boldsymbol{g}^{(k)} = \boldsymbol{0} \end{cases}$$

2. Following [Bec17, Example 9.29] we have the following update rule:

$$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \text{prox}_{s_k g}\left(\boldsymbol{x}^{(k)} - s_k\boldsymbol{A}^T\text{sgn}(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b})\right) \\ &= \mathcal{T}_{\lambda s_k}\left(\boldsymbol{x}^{(k)} - s_k\boldsymbol{A}^T\text{sgn}(\boldsymbol{A}\boldsymbol{x}^{(k)} - \boldsymbol{b})\right) \\ s_k &= \frac{1}{\|\boldsymbol{g}^{(k)}\|\sqrt{k+1}}, \quad \text{where} \quad \boldsymbol{g}^{(k)} \in \partial H(\boldsymbol{x}^{(k)}) \end{aligned}$$

[1] We rewrote function definitions so that we align with Beck's notation in Chapter 10

Here, we use the definition of the *soft thresholding* function from [Bec17, Example 6.8] with $\lambda > 0$ which is the proximal operator of the ℓ_1-norm, i.e., $\text{prox}_{\lambda\|\cdot\|_1}(\boldsymbol{x}) = \mathcal{T}_\lambda(\boldsymbol{x})$.

3. We first try to smoothen h. We know from [Bec17, Example 10.54], that for $\omega(\boldsymbol{x}) = \|\boldsymbol{x}\|_1$ the function

$$\omega_\mu(\boldsymbol{x}) = M_h^\mu(\boldsymbol{x}) = \sum_{i=1}^{n} H_\mu(x_i) \tag{2.17}$$

is a $\frac{1}{\mu}$-smooth approximation of h with parameters $(\alpha, \beta) = (1, \frac{n}{2})$, where M_h^μ is the Moreau envelope and H_μ is the Huber function.

Since $h(\boldsymbol{x}) = \omega(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$, using [Bec17, Theorem 10.46(b)], we get that $h_\mu(\boldsymbol{x})$ is $\frac{1}{\mu}$-smooth approximation of h with parameters $(\alpha, \beta) = (\|\boldsymbol{A}\|_{2,2}^2, \frac{n}{2})$.

We will write down and simplify the S-FISTA updates from [Bec17, p. 311]. But first, we know that $\boldsymbol{y}^0 = \boldsymbol{x}^0 \in \text{dom}(g) = \mathbb{R}^n$, $F_\mu = h_\mu$, $t_0 = 1$, $L_f = 0$ (since $f = 0$). From [Bec17, Theorem 10.57] we know:

$$\mu = \sqrt{\frac{\alpha}{\beta}} \frac{\varepsilon}{\sqrt{\alpha\beta}\sqrt{\alpha\beta}} = \frac{\varepsilon}{2\beta} \overset{\beta=\frac{n}{2}}{=} \frac{\varepsilon}{n} \tag{2.18}$$

$$\tilde{L} = \frac{\alpha}{\mu} = \frac{\|\boldsymbol{A}\|_{2,2}^2}{\mu} = \frac{\lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})}{\mu} \tag{2.19}$$

Lastly, using [Bec17, Theorem 6.60] we have:

$$\nabla h_\mu(\boldsymbol{x}) \overset{\boldsymbol{z}=\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}}{=} \boldsymbol{A}^T \nabla\omega(\boldsymbol{z}) = \frac{1}{\mu}\boldsymbol{A}^T(\boldsymbol{z} - \text{prox}_{\mu\|\cdot\|_1}(\boldsymbol{z})) = \frac{1}{\mu}\boldsymbol{A}^T(\boldsymbol{z} - \mathcal{T}_\mu(\boldsymbol{z})) \tag{2.20}$$

We have laid out everything we will need and we can now work on the S-FISTA updates from [Bec17, p. 311]. We will avoid referencing equations when we replace variables so that it does not get messy; everything is available above.

1. First, we compute gradient h_μ:

   The $\boldsymbol{x}^{(k+1)}$ update is:

   $$\begin{aligned} \nabla h_\mu(\boldsymbol{y}^{(k)}) &= \frac{1}{\mu}\boldsymbol{A}^T(\boldsymbol{z} - \mathcal{T}_\mu(\boldsymbol{z})) \\ \boldsymbol{z} &= \boldsymbol{A}\boldsymbol{y}^{(k)} - \boldsymbol{b} \end{aligned}$$

   Then, the $\boldsymbol{x}^{(k+1)}$ update is:

   $$\begin{aligned} \boldsymbol{x}^{(k+1)} &= \text{prox}_{\frac{1}{\tilde{L}}(\lambda g)}\left(\boldsymbol{y}^{(k)} - \frac{1}{\tilde{L}}\nabla F_\mu(\boldsymbol{y}^{(k)})\right) \\ &= \text{prox}_{\frac{\lambda}{\tilde{L}}\|\cdot\|_1}\left(\boldsymbol{y}^{(k)} - \frac{1}{\tilde{L}}\nabla h_\mu(\boldsymbol{y}^{(k)})\right) \\ &= \mathcal{T}_{\frac{\lambda}{\tilde{L}}}\left(\boldsymbol{y}^{(k)} - \frac{1}{\tilde{L}}\nabla h_\mu(\boldsymbol{y}^{(k)})\right) \end{aligned}$$

2. Then, we set: $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$

3. Lastly, we set: $\boldsymbol{y}^{(k+1)} = \boldsymbol{x}^{(k+1)} + \left(\frac{t_k - 1}{t_{k+1}}\right)(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)})$

4. Notably, we have already computed h_μ, the smooth approximation of h, in question 3. Now it is time to also smoothen (λg). Since we know that the smooth approximation of $g = ||\cdot||_1$ is ω_μ from equation (2.17), we can simply use [Bec17, Theorem 10.46(b)] and conclude that $(\lambda g_{\tilde{\mu}})$ is a $\frac{1}{\tilde{\mu}}$-smooth approximation of (λg) with parameters $(\alpha, \beta) = (\lambda, \frac{\lambda n}{2})$. Finally:

$$\nabla(\lambda g_{\tilde{\mu}})(\boldsymbol{x}) = \lambda \nabla g_{\tilde{\mu}}(\boldsymbol{x}) = \lambda \nabla \omega_{\tilde{\mu}}(\boldsymbol{x}) \overset{2.20}{=} \frac{\lambda}{\tilde{\mu}}(\boldsymbol{x} - \mathcal{T}_{\tilde{\mu}}(\boldsymbol{x})) \tag{2.21}$$

$$\tilde{\mu} \overset{2.18}{=} \frac{\varepsilon}{2\beta} \overset{\beta=\frac{\lambda n}{2}}{=} \frac{\varepsilon}{\lambda n} \tag{2.22}$$

All in all, our new $H_s = h_\mu + (\lambda g_{\tilde{\mu}})$ is a completely smooth function. We no longer need a proximal operator and are ready to write down the S-FISTA updates with no proximal step. Notably, the resulting algorithm is Nesterov Accelerated Gradient:

1. We first compute gradient H_s using the results (2.20), (2.18), (2.21) and (2.22):

$$\begin{aligned}
\nabla H_s(\boldsymbol{y}^{(k)}) &= \frac{1}{\mu}\boldsymbol{A}^T(\boldsymbol{z} - \mathcal{T}_\mu(\boldsymbol{z})) + \frac{\lambda}{\tilde{\mu}}(\boldsymbol{y}^{(k)} - \mathcal{T}_{\tilde{\mu}}(\boldsymbol{y}^{(k)})) \\
\boldsymbol{z} &= \boldsymbol{A}\boldsymbol{y}^{(k)} - \boldsymbol{b} \\
\mu &= \frac{\varepsilon}{n} \\
\tilde{\mu} &= \frac{\varepsilon}{\lambda n}
\end{aligned}$$

   Then, the $\boldsymbol{x}^{(k+1)}$ update is:

$$\begin{aligned}
\boldsymbol{x}^{(k+1)} &= \boldsymbol{y}^{(k)} - \frac{1}{L_s}\nabla H_s(\boldsymbol{y}^{(k)}) \\
L_s = L_f + L_{(\lambda g)} &\overset{(2.19)}{=} \frac{\lambda \max(\boldsymbol{A}^T\boldsymbol{A})}{\mu} + \frac{\alpha}{\tilde{\mu}} \overset{(2.22)}{=} \frac{\lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})}{\mu} + \frac{\lambda}{\tilde{\mu}}
\end{aligned}$$

2. Then, we set: $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$
3. Lastly, we set: $\boldsymbol{y}^{(k+1)} = \boldsymbol{x}^{(k+1)} + \left(\frac{t_k - 1}{t_{k+1}}\right)(\boldsymbol{x}^{(k+1)} - \boldsymbol{x}^{(k)})$

In the experiments, we construct $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{m\times n})$ and a sparse solution $\boldsymbol{x}_{\text{sol}}$ with 60% of its elements being zero. In other words, $\|\boldsymbol{x}_{\text{sol}}\|_0 \approx 0.6n$, where $\|\cdot\|_0$ is the ℓ_0-norm. Then, we set $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_{\text{sol}}$. Moreover, we set $\lambda = 0.1$ as results from previous exercises (Figure 2.3) implied that this is a solid value. Also, for both S-FISTA variants we uniformly set $\varepsilon = 100$ (after investigation). Lastly, for better comparison across algorithms, we found that setting a fixed maximum number of iterations for all methods—rather than using the stopping criterion from previous exercises—is much more informative in the plots. This approach allows us to observe and compare the convergence behavior from a more intuitive light. Naturally, the maximum iteration count is carefully tuned to ensure that no important behaviors are missed.

Case 1: $m \approx n$. This is the "balanced" case, where the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ is constrained enough to have a single solution (assuming $\boldsymbol{A}$ is full-rank). All methods are expected to perform reasonably well in this setting. As shown in Figure 2.17, subgradient descent with dynamic stepsize performs the worst among the four. Both S-FISTA variants outperform subgradient descent with Polyak's stepsize, likely due to faster convergence rates on this smooth-but-sparse objective structure. Furthermore, the classic S-FISTA performs best, winning the Nesterov's accelerated gradient.

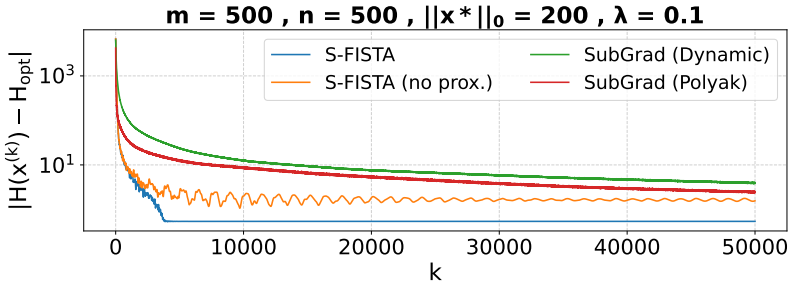

Figure 2.17: Case $m \approx n$: S-FISTA vs. S-FISTA with both functions smoothened vs. Subgradient Descent with Dynamic stepsize vs. Subgradient Descent with Polyak's stepsize. For S-FISTA we have set $\varepsilon = 0.1$. The y-axis is logarithmic

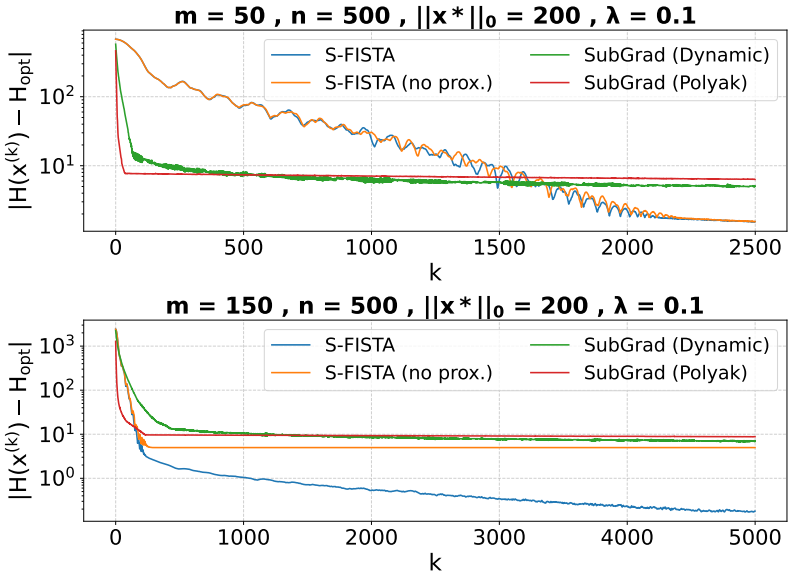

Figure 2.18: Case $m \ll n$: S-FISTA vs. S-FISTA with both functions smoothened vs. Subgradient Descent with Dynamic stepsize vs. Subgradient Descent with Polyak's stepsize. For S-FISTA we have set $\varepsilon = 0.1$. The y-axis is logarithmic

Case 2: $m \ll n$. Here, the system has infinitely many solutions and the term $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1$ cannot determine $\boldsymbol{x}$ alone, so the ℓ_1-regularization becomes plays a crucial role. Subgradient methods are expected to struggle here with slow progress. We confirm this intuition in the first two plots of Figure 2.18. This is clearly illustrated in the two plots of Figure 2.18, where S-FISTA outperforms both subgradient methods and Nesterov's accelerated algorithm.

Case 3: $m \gg n$. Here, the system is overconstrained which means that the term $\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1$ dominates the minimization. This pushes solutions toward satisfying $\boldsymbol{A}\boldsymbol{x} \approx \boldsymbol{b}$ a lot more than enforcing sparsity. We notice something very interesting: as n becomes a lot smaller than m, subgradient descent with Polyak's stepsize shows remarkably fast convergence—reaching near-optimality within very few iterations. This is evident in the plots of Figure 2.19. This aligns with the theoretical intuition that Polyak's method can leverage strong directional gradients when the residual term dominates.

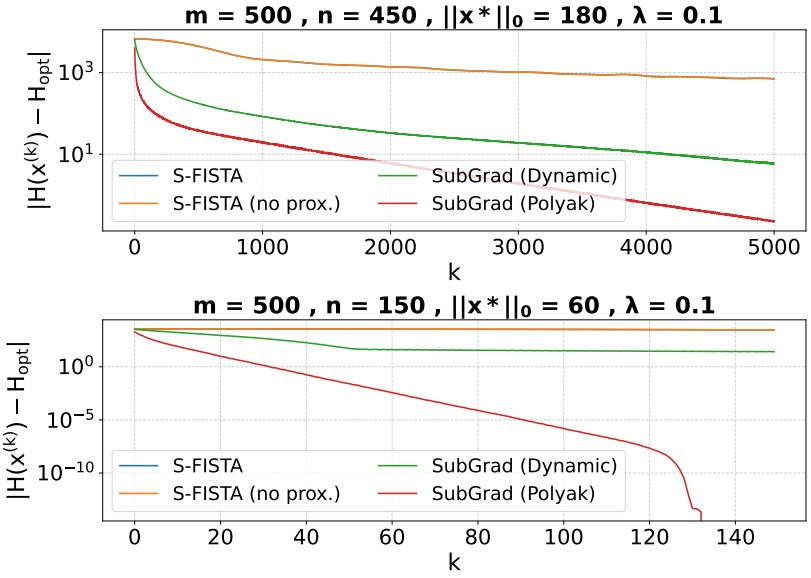

Figure 2.19: Case $m \gg n$: S-FISTA vs. S-FISTA with both functions smoothened vs. Subgradient Descent with Dynamic stepsize vs. Subgradient Descent with Polyak's stepsize. For S-FISTA we have set $\varepsilon = 0.1$. The y-axis is logarithmic

---

2.7 Matrix min ℓ_1 onto the unit simplex.

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$ $\boldsymbol{c} \in \mathbb{R}^n$, and consider the problem:

$$\min_{\boldsymbol{x}\in\mathbb{R}^n} \quad f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_1, \quad \text{subject to} \quad \boldsymbol{x} \in \Delta_n \tag{2.23}$$

1. Solve the problem via CVX and compute $\boldsymbol{x}^*$ and f_{opt}.
2. Solve the problem via the projected subgradient descent. Terminating condition: STOP if $f(\boldsymbol{x}^{(k)}) \leq c \cdot f_{\text{opt}}$. Start the algorithm from $\boldsymbol{x}^{(0)} = \frac{1}{n}\mathbf{1}$.
3. Solve the problem via mirror descent using the same terminating condition (see Example 9.19). Start the algorithm from $\boldsymbol{x}^{(0)} = \frac{1}{n}\mathbf{1}$.
4. Put in the same semilogy quantities $f(\boldsymbol{x}^{(k)}) - f_{\text{opt}}$ versus k, for both algorithms. What do you observe?
5. Experiment with (a) random $\boldsymbol{A}$ and $\boldsymbol{b}$ and (b) $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_{\text{true}} + \boldsymbol{e}$.

Solution.

1. Done ✓.

2. From exercises **2.5-2.6** we have that:

$$\boldsymbol{A}^T \text{sgn}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \in \partial f(\boldsymbol{x})$$

The projected subgradient descent has the following update rule [Bec17, p. 202, Theorem 8.28]:

$$\boldsymbol{x}^{(k+1)} = P_{\Delta_n}\left(\boldsymbol{x}^{(k)} - \frac{1}{\sqrt{k+1}\|\boldsymbol{g}^{(k)}\|_2}\boldsymbol{g}^{(k)}\right) \quad , \quad \boldsymbol{g}^{(k)} \in \partial f(\boldsymbol{x}^{(k)})$$

The projection of some $\boldsymbol{x} \in \mathbb{R}^n$ onto the unit simplex, $P_{\Delta_n}(\boldsymbol{x})$, does not have a closed-form solution. However, we follow the simple $\mathcal{O}(n \log n)$ algorithm in [WC13] in order to find it:

Algorithm 2 Projection onto the unit simplex [WC13]

Input: Initial $\boldsymbol{x} \in \mathbb{R}^d$

1: Sort $\boldsymbol{x}$ into $\boldsymbol{u}$: $u_1 \geq u_2 \geq u_3 \geq ... \geq u_n$
2: Find $\rho = \max\left(1 \leq j \leq n \ : \ u_j + \frac{1}{j}(1 - \sum_{i=1}^{j} u_i) > 0\right)$
3: Define $\lambda = \frac{1}{\rho}(1 - \sum_{i=1}^{\rho} u_i)$
4: Compute $\boldsymbol{s}$ such that $s_i = \max(x_i + \lambda, 0)$ with $i = 1, ..., n$
5: **return** $\boldsymbol{s}$ ▷ This is $P_{\Delta_n}(\boldsymbol{x})$

3. Mirror descent is the generalization of projected gradient descent. As such, if we assume that the underlying space is endowed with ℓ_2-norm, then mirror descent reduces to projected gradient descent. Thus, following the analysis of [Bec17, Chapter 9] we assume that the underlying space is endowed with ℓ_1-norm—which also means that $\|\cdot\|_* = \|\cdot\|_\infty$. Furthermore, the problem is strongly convex so $\sigma = 1$. Lastly, we choose ω as the negative cross-entropy and using the results in [Bec17, Example 9.10] we have the following updates for mirror descent:

$$\text{Compute } \boldsymbol{x}^{(k+1)} \text{ such that } \ x_i^{(k+1)} = \frac{x_i^{(k)} e^{-t_k g_i^{(k)}}}{\sum_{j=1}^{n} x_j^{(k)} e^{-t_k g_j^{(k)}}} \ \ , \ \ \boldsymbol{g}^{(k)} \in \partial f(\boldsymbol{x}^{(k)})$$

$$t_k = \frac{\sqrt{2}}{\sqrt{k+1}\|\boldsymbol{g}^{(k)}\|_\infty}$$

4-5. In the (a) random $\boldsymbol{A}$ and $\boldsymbol{b}$ experiments, we construct $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{m \times n})$ and $\boldsymbol{b} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{1})$. Conversely, in the (b) noisy $\boldsymbol{b}$ experiments we construct $\boldsymbol{A} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}_{m \times n})$, $\boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0}, \frac{1}{3} \cdot \boldsymbol{1})$ with $\boldsymbol{x}_{\text{true}} \sim \text{Dir}(\boldsymbol{1})$ and finally $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_{\text{true}} + \boldsymbol{e}$. Here, $\text{Dir}(\boldsymbol{1})$ is the multivariate Dirichlet distribution with concentration parameters $\alpha_i = 1$ for each $i = 1, ..., n$. This distribution guarantees random vectors to lie inside the unit simplex.

Case 1: $m > n$. The system might have a single unique solution to $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. When we construct an underlying ground truth by manually creating a solution inside the unit simplex, we notice that MD performs a lot better than PGD (Figure 2.20). This is because MD operates with updates entirely inside the unit simplex, whereas PGD uses subgradient steps and then projects back to the simplex which might "waste" steps. Furthermore, when we construct $\boldsymbol{A}$ and $\boldsymbol{b}$ randomly, without ensuring the solution lies inside the simplex, the differences are less defined—but still, MD performs better (Figure 2.20).

Case 2: $m \approx n$. This is the "balanced" regime, where the system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ may have a unique solution if $\boldsymbol{A}$ is full rank. In both the ground truth and random settings, MD again outperforms PGD, showing faster convergence and greater stability (Figures 2.20).

Case 3: $m < n$. The system is underdetermined with infinitely many solutions. In this regime, the simplex constraint plays a crucial role in selecting among the valid candidates. As illustrated in Figures 2.20 both the ground truth and random settings, MD consistently outperforms PGD.

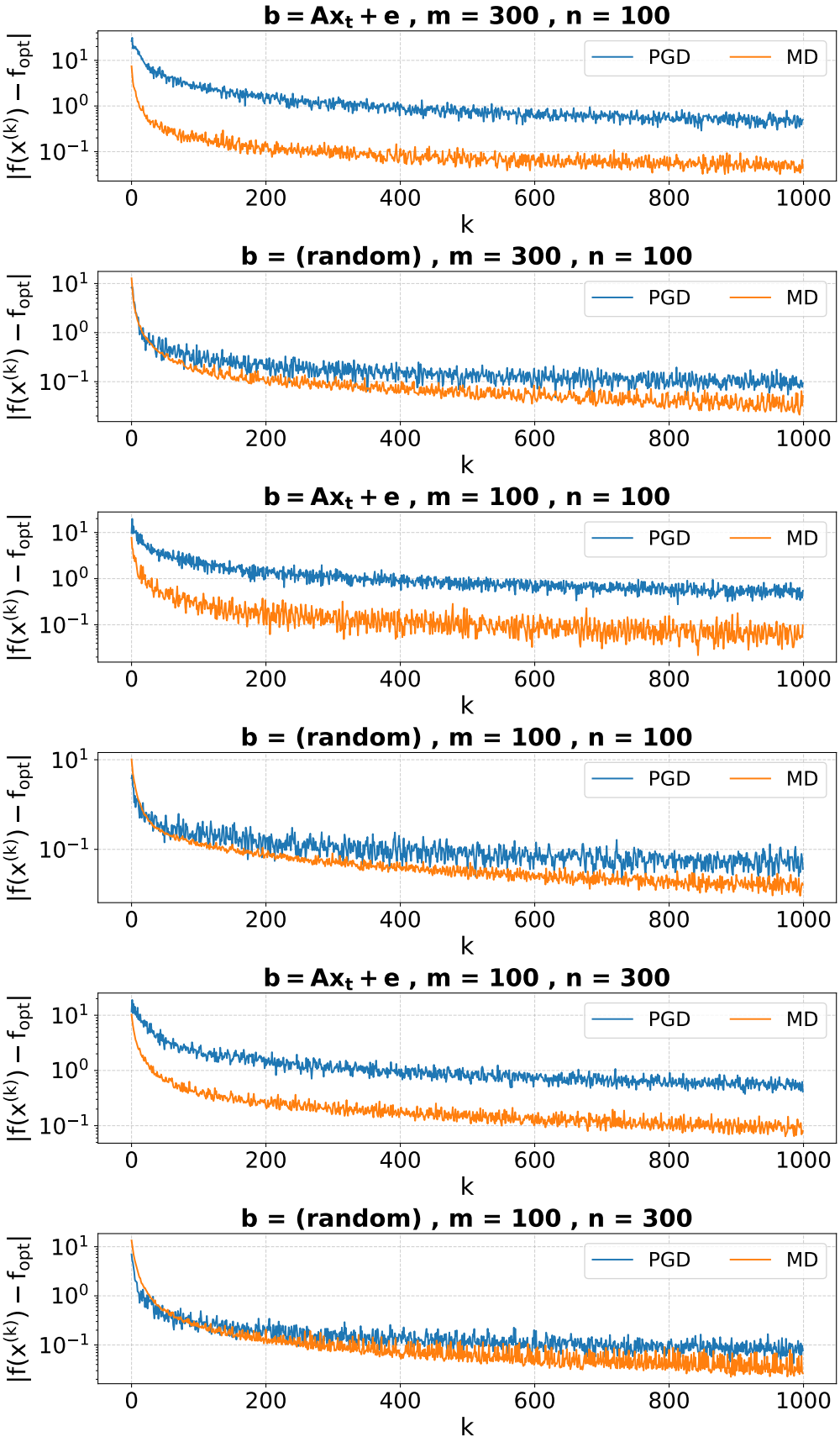

Figure 2.20: Projected Subradient Descent (PGD) vs. Mirror Descent (MD)

2.8 Matrix min l_1

Let $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ ($m \gg n$, $m \gtrapprox n$) and $\boldsymbol{b} \in \mathbb{R}^m$, and consider the problem:

$$\min_{x\in\mathbb{R}^n} \; f(\boldsymbol{x}) = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty$$

Generate $\boldsymbol{A}$ and $\boldsymbol{b}$ using (1) Gaussian distribution, and (2) uniform distribution.

1. Solve the problem via CVX and compute $\boldsymbol{x}^*$ and f_{opt}.
2. Solve the problem with the subgradient descent algorithm with dynamic stepsize. Terminating condition $f(\boldsymbol{x}^{(k)}) \leq c \cdot f_{\text{opt}}$, for $c \gtrapprox 1$. Plot $f^{(k)}_{\text{best}} - f_{\text{opt}}$ versus k.
3. Implement the subgradient descent algorithm with Polyak step (see Example 3.44). Terminating condition $f(\boldsymbol{x}^{(k)}) \leq c \cdot f_{\text{opt}}$, for $c \gtrapprox 1$ (for example, $c = 1.1, 1.01, 1.001$). Plot $f^{(k)}_{\text{best}} - f_{\text{opt}}$ versus k.
4. Put the two plots in the same figure (experiment with $m \gg n$ and $m \gtrapprox n$ and Gaussian and Uniform distributions). What do you observe?
5. Compute L_f such that $\|\boldsymbol{g}\|_2 \leq L_f$ for all $\boldsymbol{g} \in \partial f(\boldsymbol{x})$, $\boldsymbol{x} \in \mathbb{R}^n$ (see Assumption 8.12). Compute and plot the upper bound (see Theorem 8.13(c))

$$f^{(k)}_{\text{best}} - f_{\text{opt}} \leq \frac{L_f \|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\|_2}{\sqrt{k+1}}$$

   Plot the upper bound and the quantities ploted in question 4. in the same semilogy.

Solution.

1-4. Our first task is to work on defining the subgradient set $\partial f(\boldsymbol{x})$—or, at least, find some "weak result". We first define:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \vdots \\ \boldsymbol{a}_m^T \end{bmatrix} \quad , \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Then, we define the following set of indices:

$$I(\boldsymbol{x}) = \{i \in \{1, ..., m\} : |\boldsymbol{a}_i^T \boldsymbol{x} - b_i| = \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty\}$$

Now, we are ready to write down the full subgradient set [Bec17, Example 3.54]:

$$\partial f(\boldsymbol{x}) = \begin{cases} \boldsymbol{A}^T B_{\|\cdot\|_1}[\boldsymbol{0}, 1] = \boldsymbol{A}^T\{\boldsymbol{y} \in \mathbb{R}^m : \|\boldsymbol{y}\|_1 \leq 1\}, & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \\ \left\{\sum_{i\in I(\boldsymbol{x})} \lambda_i \text{sgn}(\boldsymbol{a}_i^T\boldsymbol{x} - b_i)\boldsymbol{a}_i : \sum_{i\in I(\boldsymbol{x})} \lambda_i = 1, \lambda_j \geq 0, j \in I(\boldsymbol{x})\right\}, & \boldsymbol{A}\boldsymbol{x} \neq \boldsymbol{b} \end{cases}$$

All we care about is computing some derivative $\boldsymbol{g}^{(k)}$ in the subgradient set of $f(\boldsymbol{x}^{(k)})$. Using the results above, a derivative that does the job, i.e., $\boldsymbol{g}^{(k)} \in \partial f(\boldsymbol{x}^{(k)})$, is:

$$\boldsymbol{g}^{(k)} = \begin{cases} \boldsymbol{A}^T(\frac{1}{m}\boldsymbol{1}), & \boldsymbol{A}\boldsymbol{x}^{(k)} = \boldsymbol{b} \\ \frac{1}{|I(\boldsymbol{x}^{(k)})|} \sum_{i\in I(\boldsymbol{x}^{(k)})} \text{sgn}(\boldsymbol{a}_i^T\boldsymbol{x}^{(k)} - b_i)\boldsymbol{a}_i, & \boldsymbol{A}\boldsymbol{x}^{(k)} \neq \boldsymbol{b} \end{cases} \tag{2.24}$$

The subgradient update rule is [Bec17, Equation (8.3)]:

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{x}^{(k)} - t_k \boldsymbol{g}^{(k)}, \quad \text{where } \boldsymbol{g}^{(k)} \text{ is computed using Equation (2.24)}$$

Moreover, the classic dynamic stepsize rule is [Bec17, Theorem 8.28]:

$$t_k = \begin{cases} \frac{1}{\|\boldsymbol{g}^{(k)}\|_2 \sqrt{k+1}}, & \boldsymbol{g}^{(k)} \neq \boldsymbol{0} \\ \frac{1}{L_f}, & \boldsymbol{g}^{(k)} = \boldsymbol{0} \end{cases}$$

Conversely, Polyak's stepsize rule is [Bec17, Equation (8.13)]:

$$t_k = \begin{cases} \frac{f(\boldsymbol{x}^{(k)}) - f_{\text{opt}}}{\|\boldsymbol{g}^{(k)}\|_2^2}, & \boldsymbol{g}^{(k)} \neq \boldsymbol{0} \\ 1, & \boldsymbol{g}^{(k)} = \boldsymbol{0} \end{cases}$$

We found experimentally that initialization must be handled with care: creating $\boldsymbol{A}$ and $\boldsymbol{b}$ blindly creates convergence problems—even for CVX. Thus, we follow the procedure of previous exercises and initialize $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}_{\text{sol}} + \boldsymbol{e}$ where $\boldsymbol{e}$ is some small noise from the same distribution of $\boldsymbol{A}$. It is either gaussian, $\boldsymbol{e} \sim \mathcal{N}(\boldsymbol{0}, 0.05 \cdot \boldsymbol{1})$, or uniform, $\sim \mathcal{U}_{[-0.05, 0.05]}$.

In Figure 2.21 we observe that increasing the number of rows m significantly improves convergence. As m grows relative to n, the problem becomes more overdetermined and better conditioned, which helps both algorithms to converge faster—especially the Dynamic which struggled with smaller values.

Interestingly, initializing $\boldsymbol{A}$ and $\boldsymbol{b}$ using a uniform distribution seems to result in slightly faster convergence compared to the Gaussian case, possibly due to more evenly bounded values that lead to more stable subgradients.

Furthermore, Polyak's stepsize clearly outperforms the dynamic rule, as illustrated in all plots in Figure 2.21. This makes sense: when we have access to the optimal value f_{opt}, Polyak's rule uses this information and makes much more informed step sizes to quickly zero in on the solution. This makes Polyak not just faster but also more stable and precise. However, in practice, we do not have the f_{opt} information.

5. We are asked to compute L_f such that

$$\|\boldsymbol{g}\|_2 \leq L_f, \quad \forall \boldsymbol{g} \in \partial f(\boldsymbol{x})$$

From [Bec17, Theorem 3.61], the following statements are equivalent for a convex and proper function f:

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L_f \|\boldsymbol{x} - \boldsymbol{y}\|, \ \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n \quad \Leftrightarrow \quad \|\boldsymbol{g}\|_* \leq L_f, \ \forall \boldsymbol{g} \in \partial f(\boldsymbol{x})$$

Since the dual norm of the ℓ_2-norm is again the ℓ_2-norm, bounding all subgradients in ℓ_2-norm is equivalent to showing that f is L_f-Lipschitz with respect to $\|\cdot\|_2$. Thus, to compute such an L_f, it suffices to find a constant satisfying:

$$|f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq L_f \|\boldsymbol{x} - \boldsymbol{y}\|_2, \quad \forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$$

We work as follows: let $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, then:

$$\begin{aligned}
|f(\boldsymbol{x}) - f(\boldsymbol{y})| &= |\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_\infty - \|\boldsymbol{A}\boldsymbol{y} - \boldsymbol{b}\|_\infty| \quad \text{(using the reverse triangle inequality)} \\
&\leq \|\boldsymbol{A}\boldsymbol{x} \cancel{- \boldsymbol{b}} - \boldsymbol{A}\boldsymbol{y} \cancel{+ \boldsymbol{b}}\|_\infty \\
&= \|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\|_\infty \qquad \text{(using Equation (3.1) with } q = \infty > p = 2) \\
&\leq \|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\|_2 \\
&= \|\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{y})\|_2 \qquad \text{(From [Bec17, p. 7])} \\
&\leq \underbrace{\|\boldsymbol{A}\|_{2,2}}_{L_f} \|\boldsymbol{x} - \boldsymbol{y}\|_2
\end{aligned}$$

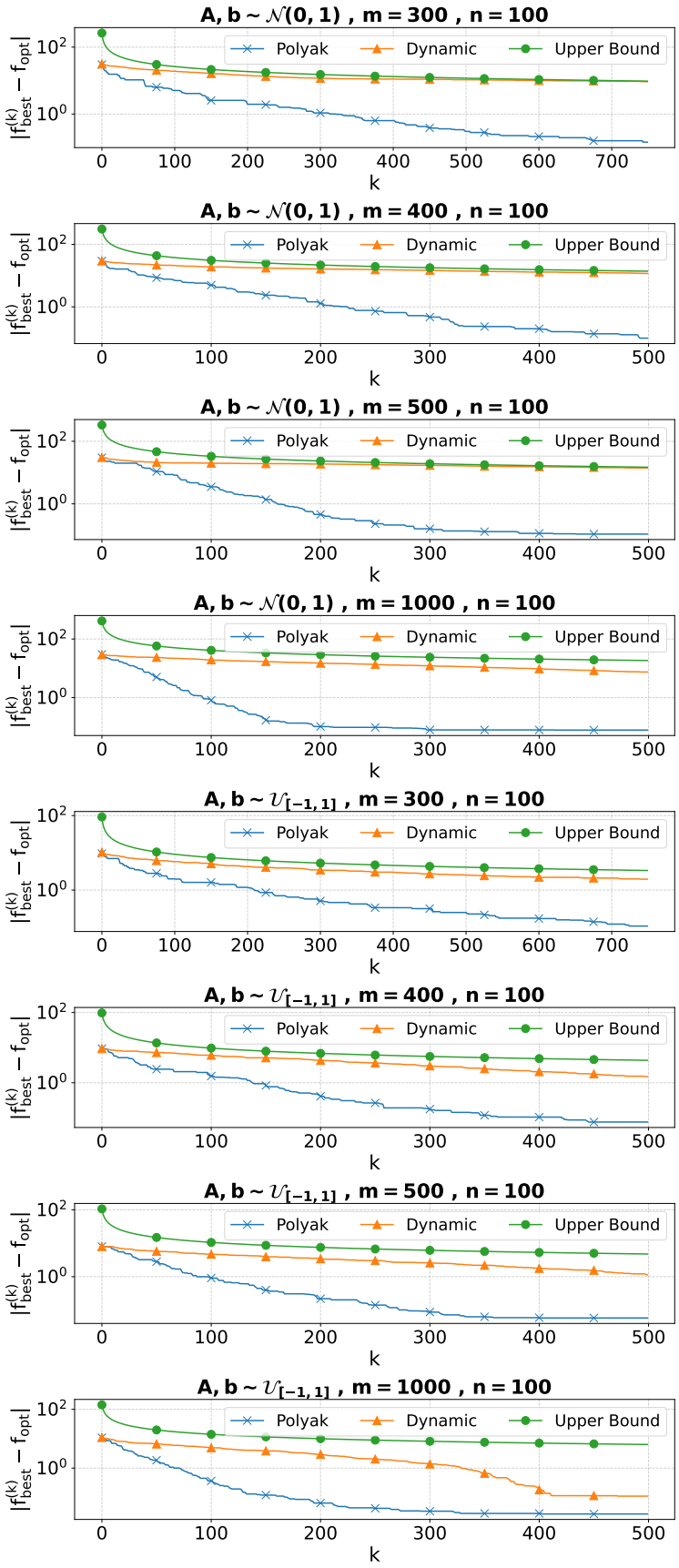

Figure 2.21: Subgradient Descent: Polyak vs. Dynamic stepsize. We also plot the Upper Bound given by the theoretical analysis and Equation (2.25). We investigate sampling $\boldsymbol{A}$ and $\boldsymbol{b}$ from either a gaussian (top four plots) or uniform (bottom four plots) distribution

All in all,

$$L_f = \|\boldsymbol{A}\|_{2,2} \overset{\text{[Bec17, Example 1.1]}}{=} \sqrt{\lambda_{\max}(\boldsymbol{A}^T\boldsymbol{A})}$$
$$UB^{(k)} = \frac{L_f\|\boldsymbol{x}^{(0)} - \boldsymbol{x}^*\|_2}{\sqrt{k+1}} \qquad \text{(Upper Bound)} \tag{2.25}$$

In Figure 2.21, we also plot the Upper Bound given by the equation above and confirm that it successfully bounds $f_{\text{best}}^{(k)} - f_{\text{opt}}$. It is especially close to the Dynamic method which inherently struggles in this problem.

2.9 Total variation denoising

Solve the total variation denoising problem (see Section 12.4.3) with various types of signals (smooth, piece wise constant) and noises.

1. solve the problems with CVX;
2. solve the ℓ_1 problem using DPG and FDPG algorithms (the special structure of matrix $\boldsymbol{D}$ makes it possible to avoid all matrix-vector multiplications and use instead, carefully chosen, vector operations.

Solution.

1-2. In this problem we are given a signal $\boldsymbol{d}$ which has been contaminated with noise:

$$\boldsymbol{d} = \boldsymbol{d}_{\text{true}} + \boldsymbol{e}$$

Our task is to denoise it. More specifically, let $\boldsymbol{d} \in \mathbb{R}^n$, $\boldsymbol{D} \in \mathbb{R}^{(n-1)\times n}$ and $\lambda > 0$. Then, our problem is the following:

$$\min_{x\in\mathbb{R}^n} \; f(\boldsymbol{x}) = \frac{1}{2}\|\boldsymbol{x} - \boldsymbol{d}\|_2^2 + \lambda\|\boldsymbol{D}\boldsymbol{x}\|_1$$

The matrix $\boldsymbol{D} \in \mathbb{R}^{(n-1)\times n}$ has the following special property:

$$\boldsymbol{D}\boldsymbol{x} = \begin{bmatrix} x_1 - x_2 \\ x_2 - x_3 \\ \vdots \\ x_{n-1} - x_n \end{bmatrix} \quad \forall \boldsymbol{x} \in \mathbb{R}^n$$

We can analytically write down this matrix $\boldsymbol{D}$:

$$\boldsymbol{D} = \begin{bmatrix} +1 & -1 & 0 & 0 & ... & 0 \\ 0 & +1 & -1 & 0 & ... & 0 \\ 0 & 0 & +1 & -1 & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & +1 & -1 \end{bmatrix}$$

Following the running example in [Bec17, Section 12.4.3], we can outline the Dual Projected Gradient (DPG) algorithm. Let the initial dual variable be $\boldsymbol{y}^{(0)} \in \mathbb{R}^{n-1}$. The k-th update step is:

$$\boldsymbol{x}^{(k)} = \boldsymbol{D}^T\boldsymbol{y}^{(k)} + \boldsymbol{d}$$
$$\boldsymbol{y}^{(k+1)} = \boldsymbol{y}^{(k)} - \frac{1}{4}\boldsymbol{D}\boldsymbol{x}^{(k)} + \frac{1}{4}\mathcal{T}_{4\lambda}(\boldsymbol{D}\boldsymbol{x}^{(k)} - 4\boldsymbol{y}^{(k)})$$

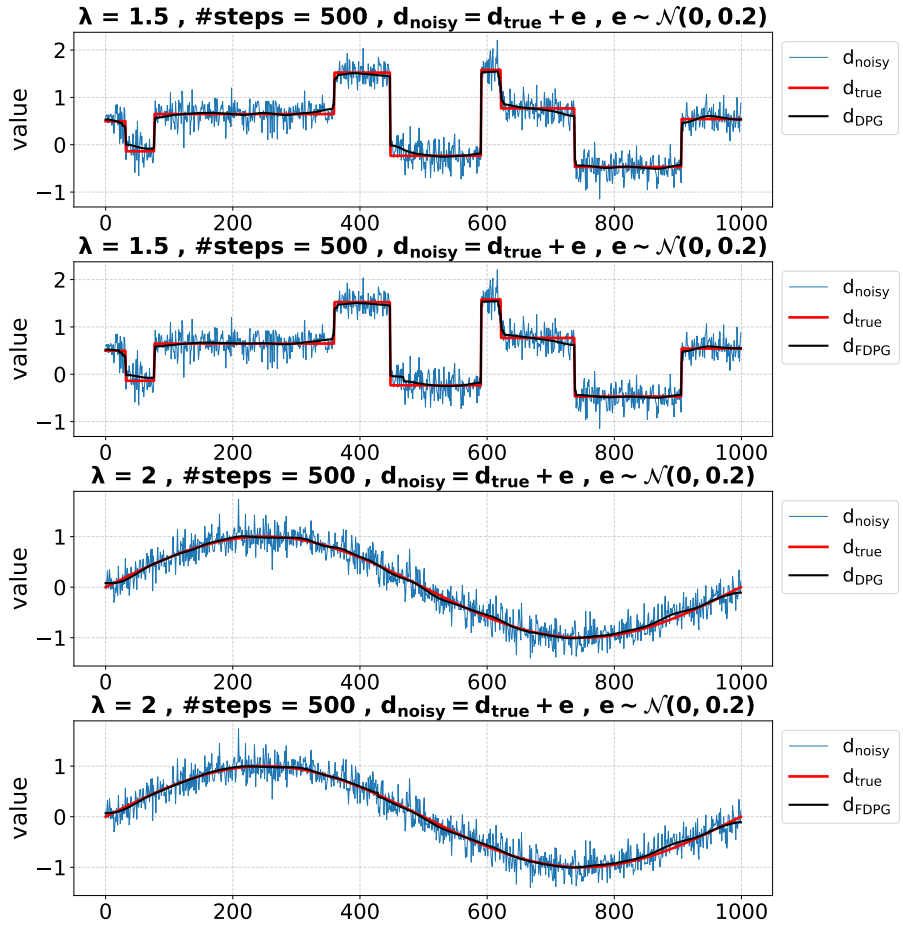

Figure 2.22: Reconstruction of contaminated signal with DPG and FDPG algorithms

$\mathcal{T}_\mu$ is the soft-thresholding operator which has already been outlined many times (2.11).

Conversely, let $\boldsymbol{w}^{(0)} = \boldsymbol{y}^{(0)} \in \mathbb{R}^{n-1}$ and $t_0 = 1$. Then, the Fast Dual Projected Gradient (FDPG) algorithm proceeds with the following updates at iteration k:

$$\boldsymbol{u}^{(k)} = \boldsymbol{D}^T \boldsymbol{w}^{(k)} + \boldsymbol{d}$$

$$\boldsymbol{y}^{(k+1)} = \boldsymbol{w}^{(k)} - \frac{1}{4}\boldsymbol{D}\boldsymbol{u}^{(k)} + \frac{1}{4}\mathcal{T}_{4\lambda}(\boldsymbol{D}\boldsymbol{u}^{(k)} - 4\boldsymbol{w}^{(k)})$$

$$t_{k+1} = \frac{1 + \sqrt{1 + 4t_k^2}}{2}$$

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{y}^{(k+1)} + \left(\frac{t_k - 1}{t_{k+1}}\right)(\boldsymbol{y}^{(k+1)} - \boldsymbol{y}^{(k)})$$

In our experiments, we construct two ground truth signals $\boldsymbol{d}_{\text{true}}$: one that is piecewise constant, and another that is smooth (specifically, a sinusoid). We then contaminate each signal with additive noise, $\boldsymbol{d}_{\text{true}} + \boldsymbol{e}$, where each component is independently drawn from one of the following distributions: $\mathcal{N}(0, 0.2)$, $\mathcal{U}_{[-0.2,0.2]}$, or Laplace$(0, 0.1)$.

These experiments are illustrated in Figures 2.22 and 2.23. We draw two main takeaways from these plots. First, FDPG and DPG exhibit very similar behavior in practice. In fact, in many reconstructions, their outputs are so visually alike that we found ourselves double-checking whether the results were numerically identical (they are not). To the naked eye, however, the differences are nearly imperceptible. Second, both algorithms manage to reconstruct the underlying signal remarkably well, even in the presence of significant noise.

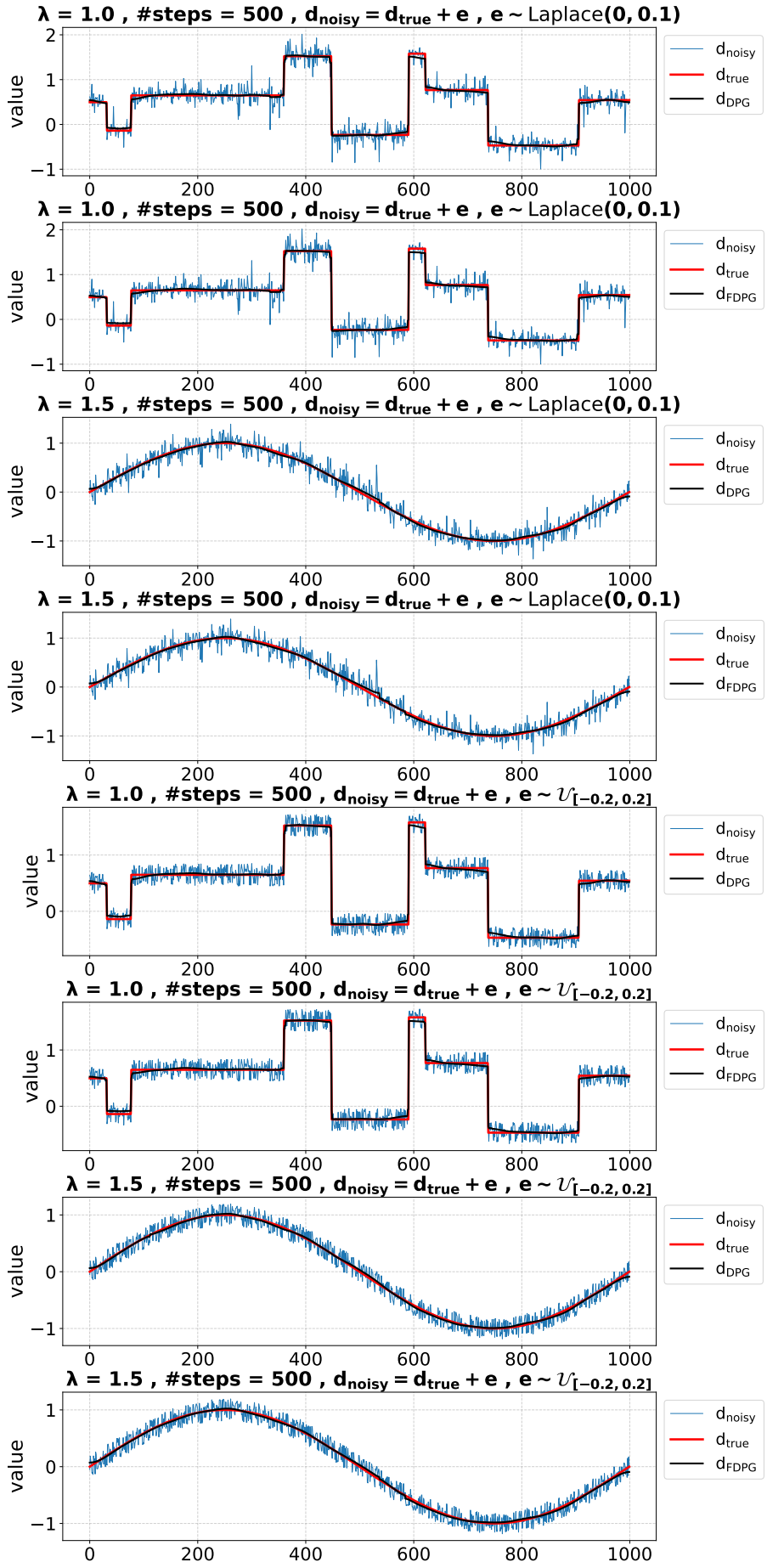

Figure 2.23: Reconstruction of contaminated signal with DPG and FDPG algorithms

2.10 Projections

1. Compute the projection onto the unit simplex (see Section 6.4.3 and Corollary 6.29). Check with CVX.
2. Compute a point of the set $S = S_1 \cap S_2$ (assuming that $S \neq \emptyset$), where
$$S_1 = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}, \quad S_2 = \mathbb{R}^n_+$$
See Example 8.23. First, check with CVX that $S \neq \emptyset$ and then implement the algorithm.
3. Let $\boldsymbol{d}, \boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^n$, with $\boldsymbol{v}_1 \leq \boldsymbol{v}_2$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$, and $\boldsymbol{b} \in \mathbb{R}^m$. Compute the projection of $\boldsymbol{d}$ onto the following sets (make sure they are nonempty):
   (a) $S_1 = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}\}$
   (b) $S_2 = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \boldsymbol{v}_1 \leq \boldsymbol{x} \leq \boldsymbol{v}_2\}$
   (c) $S_2 = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b}, \boldsymbol{v}_1 \leq \boldsymbol{x} \leq \boldsymbol{v}_2\}$

Solution.

We start by analyzing the general approach we take for this kind of feasibility problems. Assume the following formulation:

$$S = \bigcap_{i=1}^{m} C_i \neq \emptyset \tag{2.26}$$

Our task is to find some $\boldsymbol{x} \in S$. A very simple but efficient algorithm to solve this, is the *cyclic projection algorithm* [ACT19] which alternates projections in each $\boldsymbol{x}^{(k)}$ update:

$$\boldsymbol{x}^{(k+1)} = (P_{C_1} \circ P_{C_2} \circ \dots P_{C_m})(\boldsymbol{x}^{(k)}) \tag{2.27}$$

When $m = 2$ this reduces to the *alternating projection method* of [Bec17, p. 211]. Another idea is to always project onto the "most violated" constraint. In other words, on each step, we first find the set which is furthest from our current point, and project onto it. The resulting algorithm is called the *greedy projection algorithm* [Bec17, p. 210].

$$\boldsymbol{x}^{(k+1)} = P_{C_{i_k}}(\boldsymbol{x}^{(k)}) \quad \text{where} \quad i_k = \underset{i=1,\dots,m}{\operatorname{argmax}} \; d_{C_i}(\boldsymbol{x})$$

However, in order to compute i_k we need to compute m projections. Computationally, this is equivalent to actually projecting m times, i.e., one step of the cyclic algorithm of Equation (2.27). It remains unclear which is better: m projections of the cyclic method, or one greedy projection onto the most violated constraint? In the experiments we examine this question.

Furthermore, it is very helpful to make a separate analysis of the affine set $S_1 = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}\}$. The projection $P_{S_1}(\boldsymbol{x}) = \boldsymbol{x} - \boldsymbol{A}^\dagger(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})$ (see Exercise 2.3) requires the computation of the pseudo-inverse which is notoriously expensive. Thus, we seek some alternative. Let:

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1^T \\ \boldsymbol{a}_2^T \\ \vdots \\ \boldsymbol{a}_m^T \end{bmatrix} \quad , \quad \boldsymbol{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

Then, the affine set S_1 can be decomposed as the intersection of m hyperplanes:

$$S_1 = \bigcap_{i=1}^{m} H_i, \quad \text{where} \quad H_i = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}_i^T \boldsymbol{x} = b_i\}$$

This formulation allows us to use either the cyclic or the greedy algorithm where in each step we have simple vector operations (see analytical expression of the projection bellow). A nice utility of viewing this problem in this light is that any additional constraints can be integrated into equation (2.26) without changing much—we simply add another set.

Moving on, in each question we decompose the sets following the above analysis. But first, we provide some well-known analytical solutions to projections onto different sets. These will be very useful later on.

- Projection onto hyperplane $S = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}^T \boldsymbol{x} = b\}$ [ACT19]:

$$P_S(\boldsymbol{x}) = \boldsymbol{x} - \frac{\boldsymbol{a}^T \boldsymbol{x} - b}{\|\boldsymbol{a}\|_2^2} \boldsymbol{a}$$

- Projection onto half-space $S = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}^T \boldsymbol{x} \leq b\}$ [ACT19]:

$$P_S(\boldsymbol{x}) = \boldsymbol{x} + \frac{\max\{0, b - \boldsymbol{a}^T \boldsymbol{x}\}}{\|\boldsymbol{a}\|_2^2} \boldsymbol{a}$$

- Projection onto a box $S = \text{Box}[\boldsymbol{v}_1, \boldsymbol{v}_2] = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{v}_1 \leq \boldsymbol{x} \leq \boldsymbol{v}_2\}$ [BV04, p. 399]:

$$P_S(\boldsymbol{x}) = \min\{\max\{\boldsymbol{x}, \boldsymbol{v}_1\}, \boldsymbol{v}_2\} \quad \text{(piecewise)}$$

- Projection onto the nonnegative orthant $S = \mathbb{R}_+^n = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \geq \boldsymbol{0}\}$ [BV04, p. 399]:

$$P_S(\boldsymbol{x}) = \max\{\boldsymbol{x}, \boldsymbol{0}\} \quad \text{(piecewise)}$$

1. We decompose the set Δ_n as an intersection between a hyperplane and the nonnegative orthant:

$$\begin{aligned} &S = C_1 \cap C_2 \\ &C_1 = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{1}^T \boldsymbol{x} = 1\} \\ &C_2 = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \geq \boldsymbol{0}\} \end{aligned}$$

2. We decompose S into the intersection of m hyperplanes and the nonnegative orthant:

$$\begin{aligned} &S = \bigcap_{i=1}^{m+1} C_i \\ &C_i = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}_i^T \boldsymbol{x} = b_i\}, \quad i = 1, ..., m \\ &C_{m+1} = \mathbb{R}_+^n \end{aligned}$$

3(a). We decompose S into the intersection of m half-spaces:

$$\begin{aligned} &S = \bigcap_{i=1}^{m} C_i \\ &C_i = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}_i^T \boldsymbol{x} \leq b_i\}, \quad i = 1, ..., m \end{aligned}$$

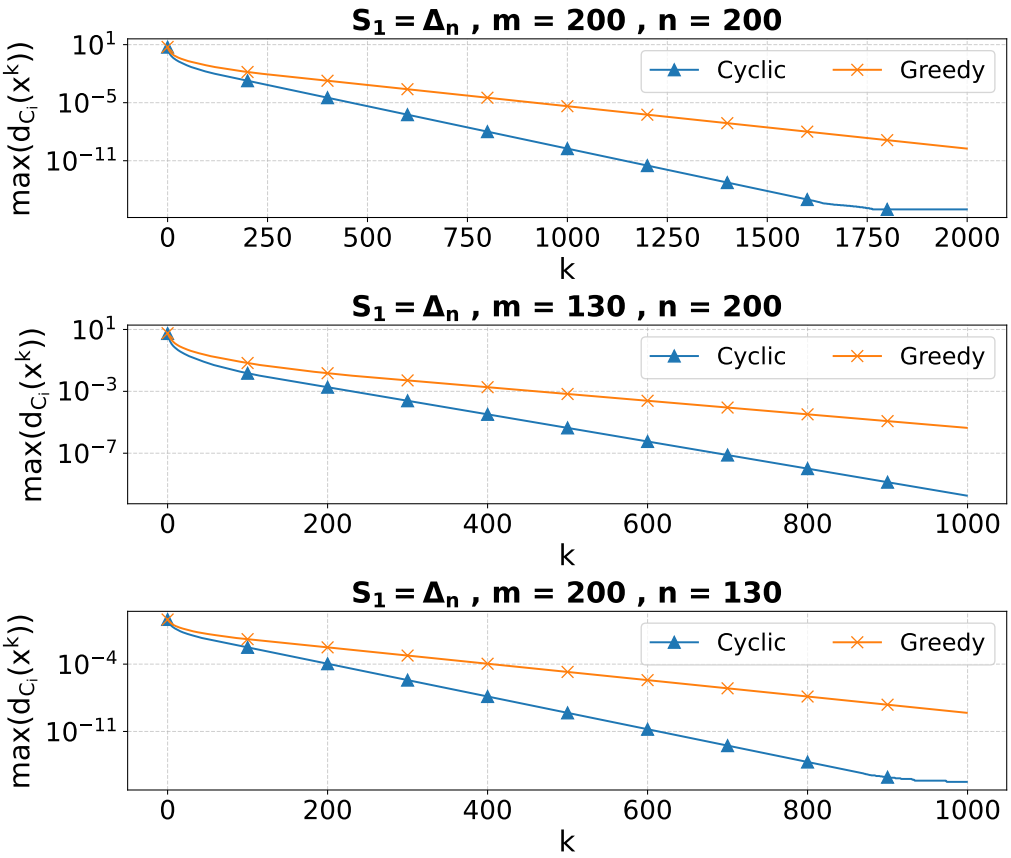

Figure 2.24: Cyclic Projection Algorithm vs. Greedy Projection Algorithm

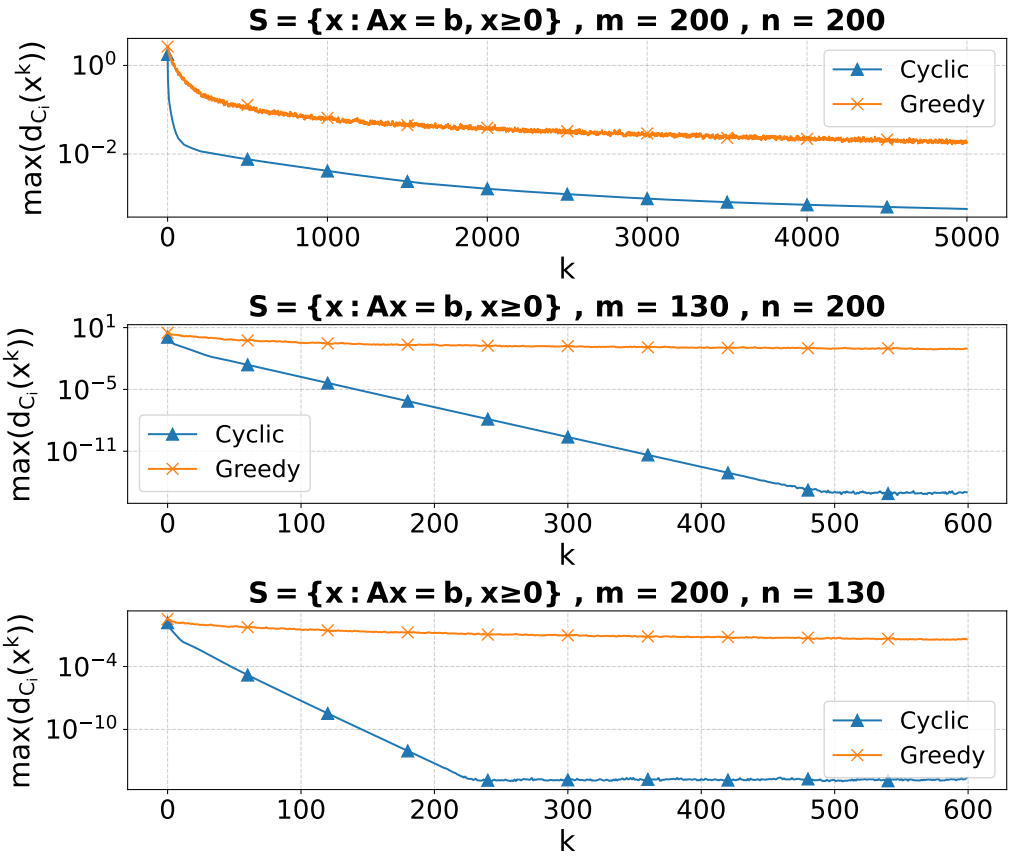

Figure 2.25: Cyclic Projection Algorithm vs. Greedy Projection Algorithm

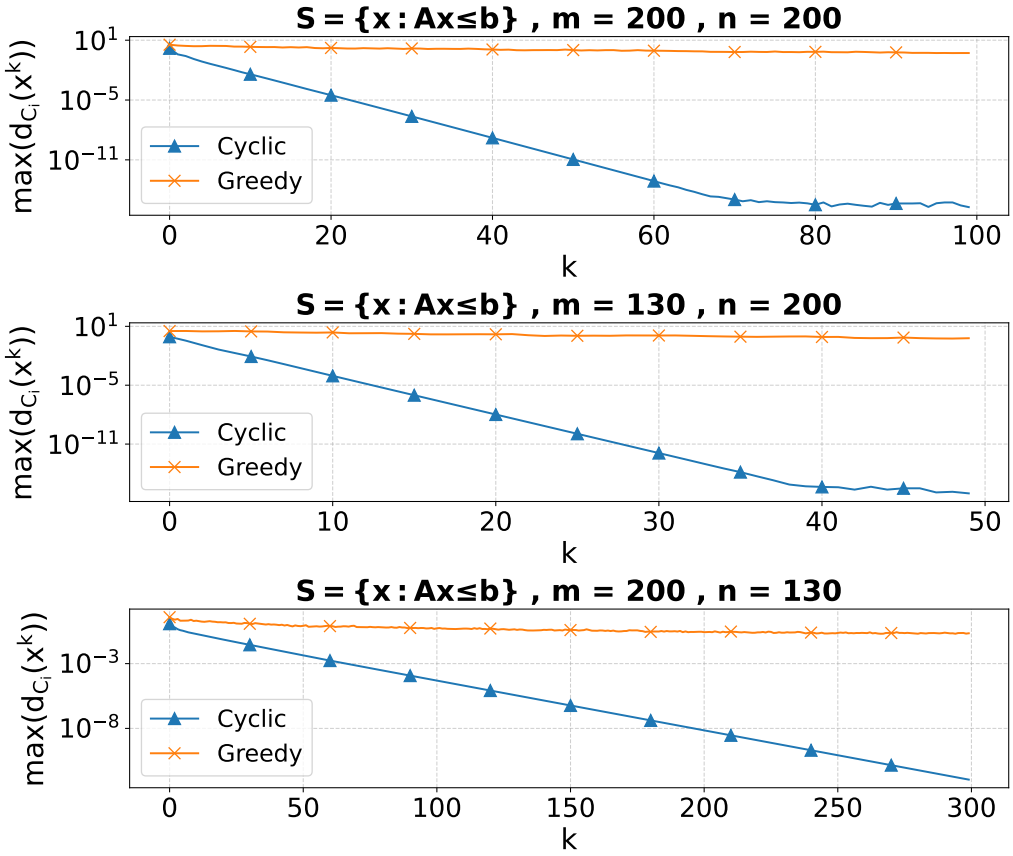

Figure 2.26: Cyclic Projection Algorithm vs. Greedy Projection Algorithm

3(b). We decompose S into the intersection of m hyper-planes and a box:

$$S = \bigcap_{i=1}^{m+1} C_i$$
$$C_i = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}_i^T \boldsymbol{x} = b_i\}, \quad i = 1, ..., m$$
$$C_{m+1} = \text{Box}[\boldsymbol{v}_1, \boldsymbol{v}_2]$$

3(c). We decompose S into the intersection of m half-spaces and a box:

$$S = \bigcap_{i=1}^{m+1} C_i$$
$$C_i = \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{a}_i^T \boldsymbol{x} \leq b_i\}, \quad i = 1, ..., m$$
$$C_{m+1} = \text{Box}[\boldsymbol{v}_1, \boldsymbol{v}_2]$$

We empirically observe (Figures 2.24-2.28) that the cyclic projection method consistently and significantly outperforms the greedy strategy across all experiments. While the greedy algorithm selects the most violated constraint at each iteration, doing so requires computing all distances to the constraint sets—making one greedy step equivalent to m cyclic steps. Furthermore, the greedy method does not achieve faster convergence; in fact, the cyclic method converges substantially faster both in terms of iteration count. The difference is often staggering, with cyclic projection reaching near-feasibility way before the greedy method. These results suggest that, in practice, greedy projection is rarely worth the cost: it is more efficient to perform m projections cyclically than to compute the maximum distance and project once onto the most violated set.

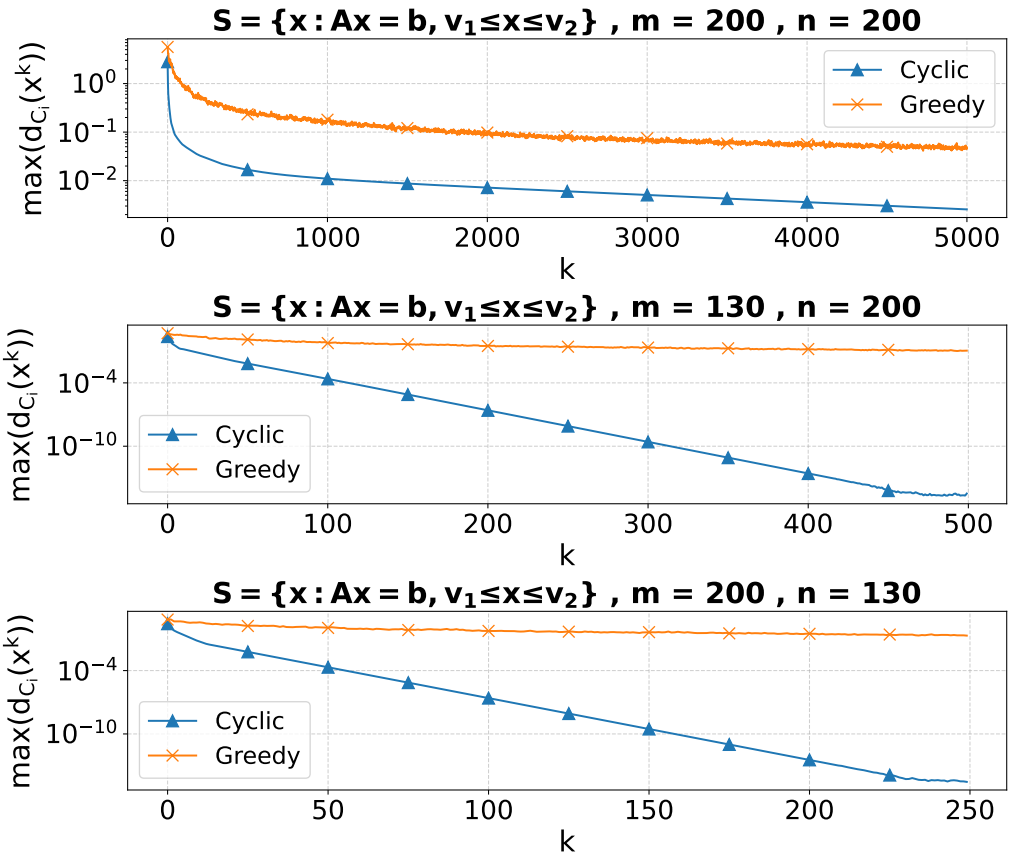

Figure 2.27: Cyclic Projection Algorithm vs. Greedy Projection Algorithm

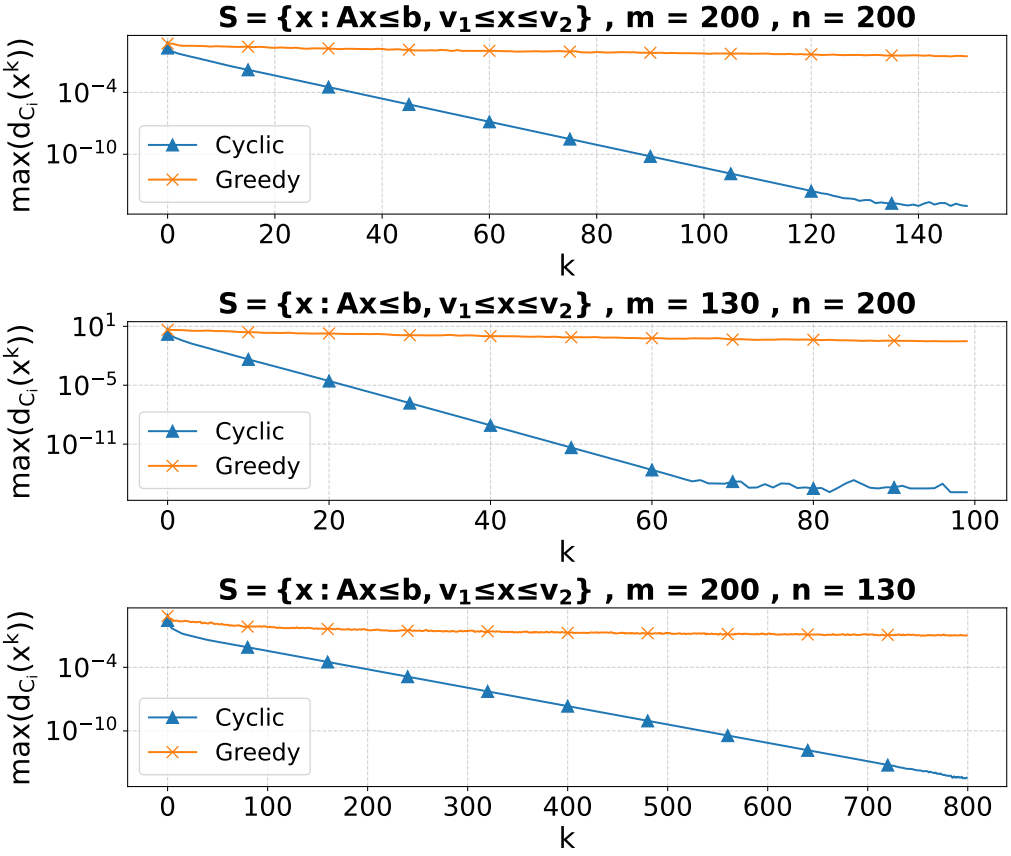

Figure 2.28: Cyclic Projection Algorithm vs. Greedy Projection Algorithm

3. UW–Madison: CS 726, HW #1

Q1. All λ_p norms are related via the following inequalities:

$$\forall\ q > p \geq 1\ ,\ \forall\ \boldsymbol{x} \in \mathbb{R}^d \quad \|\boldsymbol{x}\|_q \leq \|\boldsymbol{x}\|_p \leq d^{\frac{1}{p}-\frac{1}{q}}\|\boldsymbol{x}\|_q \tag{3.1}$$

Provide examples of non-zero vectors (vectors whose elements are not all zeros) for which these inequalities are tight (satisfied with equality).

Solution. We have the following conditions on tightness:

1. $\|\boldsymbol{x}\|_q \leq \|\boldsymbol{x}\|_p$ becomes an equality when all components of $\boldsymbol{x}$ are equal in absolute value, or all but one are zero. For example, $\boldsymbol{x} = \alpha\mathbf{1}$ or $\boldsymbol{x} = \boldsymbol{e}_i$ (for some $i \in [d]$).

2. $\|\boldsymbol{x}\|_p \leq d^{\frac{1}{p}-\frac{1}{q}}\|\boldsymbol{x}\|_q$ becomes an equality when all components of $\boldsymbol{x}$ are equal in absolute value. For example, $\boldsymbol{x} = \begin{bmatrix}\alpha & -\alpha & \alpha & \dots\end{bmatrix}$ which means $\|\boldsymbol{x}\|_\mu = d^{\frac{1}{\mu}}|\alpha|$.

Q2. Let p, q such that $p \geq 1$ and $\frac{1}{q} + \frac{1}{p} = 1$. Suppose that you are given a differentiable function $f : \mathbb{R}^d \to \overline{\mathbb{R}}$, a number $p \geq 1$, and a constant L_p such that:

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d \quad \|\nabla f(\boldsymbol{x}) + \nabla f(\boldsymbol{y})\|_q \leq L_p\|\boldsymbol{x} - \boldsymbol{y}\|_p \tag{3.2}$$

where $\nabla f(\boldsymbol{x})$ denotes the gradient (the fector of partial derivatives) of f at $\boldsymbol{x}$. What is the smallest constant L_2 (as a function of p, d and L_p) for which the following holds:

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d \quad \|\nabla f(\boldsymbol{x}) + \nabla f(\boldsymbol{y})\|_2 \leq L_2\|\boldsymbol{x} - \boldsymbol{y}\|_2 \tag{3.3}$$

How large can L_2 be in the worst case (over the possible values of p)?

Solution. We begin by re-writing the inequalities of **Q1** in the following equivalent way:

$$\begin{gathered}\forall\ r > s \geq 1\ ,\ \forall\ \boldsymbol{x} \in \mathbb{R}^d \quad \|\boldsymbol{x}\|_r \leq \|\boldsymbol{x}\|_s \leq d^{\frac{1}{s}-\frac{1}{r}}\|\boldsymbol{x}\|_r \\ \Leftrightarrow \\ \forall\ r, s \geq 1\ ,\ \forall\ \boldsymbol{x} \in \mathbb{R}^d \quad \|\boldsymbol{x}\|_s \leq d^{\max\{0,\frac{1}{s}-\frac{1}{r}\}}\|\boldsymbol{x}\|_r\end{gathered}$$

We get the following two inequalities by first setting $s = 2$, and then $r = 2$:

$$\forall\ r \geq 1\ ,\ \forall\ \boldsymbol{x} \in \mathbb{R}^d \quad \|\boldsymbol{x}\|_2 \leq d^{\max\{0,\frac{1}{2}-\frac{1}{r}\}}\|\boldsymbol{x}\|_r \tag{3.4}$$

$$\forall\ s \geq 1\ ,\ \forall\ \boldsymbol{x} \in \mathbb{R}^d \quad \|\boldsymbol{x}\|_s \leq d^{\max\{0,\frac{1}{s}-\frac{1}{2}\}}\|\boldsymbol{x}\|_2 \tag{3.5}$$

We proceed as follows:

$$\begin{aligned}
\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d \quad \|\nabla f(\boldsymbol{x}) + \nabla f(\boldsymbol{y})\|_2 &\overset{(3.4)}{\leq} d^{\max\{0, \frac{1}{2}-\frac{1}{q}\}} \|\nabla f(\boldsymbol{x}) + \nabla f(\boldsymbol{y})\|_q \\
&\overset{(3.2)}{\leq} d^{\max\{0, \frac{1}{2}-\frac{1}{q}\}} L_p \|\boldsymbol{x} - \boldsymbol{y}\|_p \\
&\overset{(3.5)}{\leq} d^{\max\{0, \frac{1}{2}-\frac{1}{q}\}} L_p d^{\max\{0, \frac{1}{p}-\frac{1}{2}\}} \|\boldsymbol{x} - \boldsymbol{y}\|_2 \\
&\overset{\frac{1}{2}-\frac{1}{q}=\frac{1}{p}-\frac{1}{2}}{=} d^{\max\{0, \frac{1}{p}-\frac{1}{2}\}} d^{\max\{0, \frac{1}{p}-\frac{1}{2}\}} L_p \|\boldsymbol{x} - \boldsymbol{y}\|_2 \\
&= d^{\max\{0, \frac{2}{p}-1\}} L_p \|\boldsymbol{x} - \boldsymbol{y}\|_2
\end{aligned}$$

So, we have shown the following:

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d \quad \|\nabla f(\boldsymbol{x}) + \nabla f(\boldsymbol{y})\|_2 \leq d^{\max\{0, \frac{2}{p}-1\}} L_p \|\boldsymbol{x} - \boldsymbol{y}\|_2 \tag{3.6}$$

Given that both Equation (3.6)—which we have just proven—and Equation (3.3) hold for every $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^d$ it must be that:

$$L_2 = d^{\max\{0, \frac{2}{p}-1\}} L_p = \begin{cases} L_p & , \quad p \geq 2 \\ dL_p & , \quad p = 1 \end{cases}$$

Of course, the largest value of L_2 is dL_p in the case of $p = 1$.

Q3. Let $\mathcal{X} \subseteq \mathbb{R}^d$ and $\mathcal{Y} \subseteq \mathbb{R}^d$ be convex sets.

(i) Prove that $\mathcal{X} \cap \mathcal{Y}$ is convex.

(ii) Provide an example that shows that $\mathcal{X} \cup \mathcal{Y}$ is not necessarily convex.

Solution. Let $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathcal{X} \cap \mathcal{Y}$ and $t \in [0, 1]$. Of course, $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathcal{X}$ and $\boldsymbol{z}_1, \boldsymbol{z}_2 \in \mathcal{Y}$. From the convexity of the two sets, we get $t\boldsymbol{z}_1 + (1-t)\boldsymbol{z}_2 \in \mathcal{X}$ and $t\boldsymbol{z}_1 + (1-t)\boldsymbol{z}_2 \in \mathcal{Y}$ which by definition means $t\boldsymbol{z}_1 + (1-t)\boldsymbol{z}_2 \in \mathcal{X} \cap \mathcal{Y}$, proving that $\mathcal{X} \cap \mathcal{Y}$ is convex. Set intersection preserves convexity. In contrast, set union does not preserve convexity; simply take $\mathcal{X} = \{0\}$ and $\mathcal{Y} = \{1\}$.

Q4. (Jensen's Inequality). Let $f : \mathbb{R}^d \to \overline{\mathbb{R}}$ be a convex function. Prove that for any sequence of vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_k \in \mathbb{R}^d$ and any sequence of non-negative scalars $\alpha_1, \alpha_2, ..., \alpha_k \geq 0$ such that $\sum_{i=1}^k \alpha_i = 1$ we have:

$$f\left(\sum_{i=1}^k \alpha_i \boldsymbol{x}_i\right) \leq \sum_{i=1}^k \alpha_i f(\boldsymbol{x}_i) \tag{3.7}$$

Solution. This can be proven by induction. Take $k = 2$, then, take any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$ and $\alpha_1, \alpha_2 \in \mathbb{R}$ with $\alpha_1 + \alpha_2 = 1$. Then, equation (3.7) becomes

$$f\left(\alpha_1 \boldsymbol{x}_1 + (1-\alpha_1)\boldsymbol{x}_2\right) \leq \alpha_1 f(\boldsymbol{x}_1) - (1-\alpha_1) f(\boldsymbol{x}_2)$$

which holds by definition of the convexity of f. We have established that the proposition holds for $k = 2$. Now, we assume the proposition holds for some n. Then, we consider $n+1$ with $\alpha_1, \alpha_2, ..., \alpha_{n+1} \geq 0$ and $\sum_{i=1}^{n+1} \alpha_i = 1$ and proceed as follows:

$$f\left(\sum_{i=1}^{n+1} \alpha_i \boldsymbol{x}_i\right) = f\left(\alpha_{n+1}\boldsymbol{x}_{n+1} + \sum_{i=1}^{n} \alpha_i \boldsymbol{x}_i\right) = f\Bigg(\alpha_{n+1}\boldsymbol{x}_{n+1} + (1-\alpha_{n+1}) \underbrace{\sum_{i=1}^{n} \frac{\alpha_i}{1-\alpha_{n+1}} \boldsymbol{x}_i}_{\boldsymbol{x}_s}\Bigg)$$

$$= f\left(\alpha_{n+1}\boldsymbol{x}_{n+1} + (1-\alpha_{n+1})\boldsymbol{x}_s\right) \overset{\text{(conv.)}}{\leq} \alpha_{n+1} f\left(\boldsymbol{x}_{n+1}\right) + (1-\alpha_{n+1}) f\left(\boldsymbol{x}_s\right)$$

$$= \alpha_{n+1} f\left(\boldsymbol{x}_{n+1}\right) + (1-\alpha_{n+1}) f\left(\sum_{i=1}^{n} \frac{\alpha_i}{1-\alpha_{n+1}} \boldsymbol{x}_i\right)$$

$$\overset{*}{\leq} \alpha_{n+1} f\left(\boldsymbol{x}_{n+1}\right) + \cancel{(1-\alpha_{n+1})} \sum_{i=1}^{n} \frac{\alpha_i}{\cancel{1-\alpha_{n+1}}} f\left(\boldsymbol{x}_i\right) = \sum_{i=1}^{n+1} \alpha_i f(\boldsymbol{x}_i) \tag{3.8}$$

It is important to clarify why the inequality with the asterisk * holds. First, we have assumed that $\sum_{i=1}^{n+1} \alpha_i = 1$ which means that $\sum_{i=1}^{n} \alpha_i = 1 - \alpha_{n+1}$. Then, notice that $\sum_{i=1}^{n} \frac{\alpha_i}{1-\alpha_{n+1}} = \frac{\sum_{i=1}^{n} \alpha_i}{1-\alpha_{n+1}} = \frac{1-\alpha_{n+1}}{1-\alpha_{n+1}} = 1$. Also, $\frac{\alpha_i}{1-\alpha_{n+1}} \geq 0 \; \forall i \in [n]$. We assumed that the proposition holds for n and we just showed that $\sum_{i=1}^{n} \frac{\alpha_i}{1-\alpha_{n+1}} = 1$ and $\frac{\alpha_i}{1-\alpha_{n+1}} \geq 0$ which means that these coefficients follow the requirements of the proposition, which means it holds for them too, that is, $f\left(\sum_{i=1}^{n} \frac{\alpha_i}{1-\alpha_{n+1}} \boldsymbol{x}_i\right) \leq \sum_{i=1}^{n} \frac{\alpha_i}{1-\alpha_{n+1}} f\left(\boldsymbol{x}_i\right)$.

In equation (3.8) we proved that the proposition holds for $k = n+1$, given that it holds for $k = n$, which by induction (it holds for $k = 2$) implies that it holds for every k.

Q6. Let $f : \mathbb{R}^d \to \overline{\mathbb{R}}$. Prove that f is convex if and only if its epigraph, defined as:

$$\text{epi}(f) = \{(\boldsymbol{x}, a) : \boldsymbol{x} \in \mathbb{R}^d, a \in \mathbb{R}, f(\boldsymbol{x}) \leq a\}$$

is convex.

Solution. We will first prove: epi(f) is convex $\Rightarrow$ f is convex. Thus, we assume that epi(f) is convex. We take some points $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d$ and some $\lambda \in [0, 1]$. Notice that the points $(\boldsymbol{x}_1, f(\boldsymbol{x}_1)), (\boldsymbol{x}_2, f(\boldsymbol{x}_2)) \in \text{epi}(f)$. Since epi($f$) is convex we have:

$$(1-\lambda)\left(\boldsymbol{x}_1, f(\boldsymbol{x}_1)\right) + \left(\boldsymbol{x}_2, f(\boldsymbol{x}_2)\right) \in \text{epi}(f) \quad \Rightarrow$$

$$\left((1-\lambda)\boldsymbol{x}_1 + \lambda\boldsymbol{x}_2, (1-\lambda)f(\boldsymbol{x}_1) + \lambda f(\boldsymbol{x}_2)\right) \in \text{epi}(f) \quad \overset{\text{epi def.}}{\Rightarrow}$$

$$f\left((1-\lambda)\boldsymbol{x}_1 + \lambda\boldsymbol{x}_2\right) \leq (1-\lambda)f(\boldsymbol{x}_1) + \lambda f(\boldsymbol{x}_2) \quad \overset{\text{conv. def.}}{\Rightarrow} f \text{ is convex}$$

$$f((1-\lambda)\boldsymbol{x}_1 + \lambda\boldsymbol{x}_2) \overset{\text{Convexity}}{\leq} (1-\lambda)f(\boldsymbol{x}_1) + \lambda f(\boldsymbol{x}_2) \quad \Rightarrow$$

$$f((1-\lambda)\boldsymbol{x}_1 + \lambda\boldsymbol{x}_2) \leq (1-\lambda)a_1 + \lambda a_2 \quad \overset{\text{epi def.}}{\Rightarrow}$$

$$((1-\lambda)\boldsymbol{x}_1 + \lambda\boldsymbol{x}_2, (1-\lambda)f(\boldsymbol{x}_1) + \lambda f(\boldsymbol{x}_2)) \in \text{epi}(f) \quad \overset{\text{conv. def.}}{\Rightarrow} \text{epi}(f) \text{ is convex}$$

Q7. Let $\boldsymbol{A} \in \mathbb{S}^d$ with eigenvalues $\lambda_1 \leq \lambda_2 \leq ... \leq \lambda_d$. Prove that, $\forall\ \boldsymbol{x} \in \mathbb{R}^d$:

(i) $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \geq \lambda_1\|\boldsymbol{x}\|_2^2$.

(ii) $\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \leq \lambda_d\|\boldsymbol{x}\|_2^2$.

Solution. From [Str21, Spectral Theorem] we can write every symmetric matrix $\boldsymbol{A} \in \mathbb{S}^d$ as

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1} \quad , \quad \boldsymbol{Q}^{-1} = \boldsymbol{Q}^T$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix with $\Lambda_{i,i} = \lambda_i$ and $\boldsymbol{Q}$ is an orthogonal matrix with the corresponding eigenvectors as columns. From all this we get:

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}\boldsymbol{x} = (\boldsymbol{Q}^T\boldsymbol{x})^T\boldsymbol{\Lambda}\underbrace{\boldsymbol{Q}^T\boldsymbol{x}}_{\boldsymbol{v}} = \boldsymbol{v}^T\boldsymbol{A}\boldsymbol{v} = \sum_{i,j} v_i\Lambda_{i,j}v_j =$$

$$= \sum_i v_i\Lambda_{i,i}v_i = \sum_i \lambda_i v_i^2 \quad \begin{cases} \geq \sum_i \lambda_1 v_i^2 = \lambda_1\|\boldsymbol{v}\|_2^2 = \lambda_1\|\boldsymbol{x}\|_2^2 \\ \leq \sum_i \lambda_d v_i^2 = \lambda_d\|\boldsymbol{v}\|_2^2 = \lambda_d\|\boldsymbol{x}\|_2^2 \end{cases}$$

Of course, at the end, we used the fact that:

$$\|\boldsymbol{v}\|_2^2 = \boldsymbol{v}^T\boldsymbol{v} = (\boldsymbol{Q}^T\boldsymbol{x})^T\boldsymbol{Q}^T\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{Q}^T\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{I}\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{x} = \|\boldsymbol{x}\|_2^2$$

Q8. Let $\boldsymbol{A} \in \mathbb{R}^{d\times d}$ defined by: $A_{i,i} = 2$ for $1 \leq i \leq d$, $A_{i,i+1} = A_{i+1,i} = -1$, for $1 \leq i \leq d-1$ and $A_{d,1} = A_{1,d} = -1$. That is, $\boldsymbol{A}$ is defined as:

$$\boldsymbol{A} = \begin{bmatrix} 2 & -1 & 0 & 0 & \dots & 0 & -1 \\ -1 & 2 & -1 & 0 & \dots & 0 & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 & 0 \\ 0 & 0 & -1 & 2 & -1 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ -1 & 0 & 0 & 0 & \dots & -1 & 2 \end{bmatrix}$$

Is $\boldsymbol{A}$ positive semidefinite (PSD)? What is its smallest eigenvalue? Justify your answers.

Solution. Let $\boldsymbol{x} \in \mathbb{R}^d$. We will decompose the analytical sum of the quadratic form in three parts: (1) the diagonal of 2's, (2) the two equal "diagonals" of -1's, and (3) the two

lone -1's.

$$\begin{aligned}
\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} &= \sum_{i,j} x_i A_{i,j} x_j = \sum_{i=1}^{d} x_i A_{i,i} x_i + 2\sum_{i=1}^{d-1} x_{i+1} A_{i+1,i} x_i - 2x_1 x_d = \\
&= 2\sum_{i=1}^{d} x_i^2 - 2\sum_{i=1}^{d-1} x_i x_{i+1} - 2x_1 x_d = \\
&= -2x_1 x_d + \sum_{i=1}^{d} x_i^2 + \sum_{j=1}^{d} x_j^2 - 2\sum_{i=1}^{d-1} x_i x_{i+1} = \\
&= x_1^2 - 2x_1 x_d + x_d^2 + \sum_{i=2}^{d} x_i^2 + \sum_{j=1}^{d-1} x_j^2 - 2\sum_{i=1}^{d-1} x_i x_{i+1} = \\
&= (x_1 - x_d)^2 + \sum_{i=1}^{d-1} x_{i+1}^2 + x_i^2 - 2x_i x_{i+1} = \\
&= (x_1 - x_d)^2 + \sum_{i=1}^{d-1} (x_{i+1} - x_i)^2 \geq 0 \qquad (3.9)
\end{aligned}$$

We proved that $\boldsymbol{A}$ is positive semidefinite. Let λ_1 be the smallest eigenvalue of $\boldsymbol{A}$. We take $\boldsymbol{x} = \mathbf{1}$ and use the result that we proved in question **Q7** with equation (3.9):

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} \overset{(3.9)}{=} 0 \geq \lambda_1 \|\boldsymbol{x}\|_2^2 = \lambda_1 d \quad \Longrightarrow \quad \lambda_1 \leq 0$$

Since $\boldsymbol{A}$ is positive semidefinite all the eigenvalues are nonnegative which means that it can only be that $\lambda_1 = 0$.

A. Notation

A.1 Multivariate Calculus & Beyond

Definition 1 *Let $f : \mathbb{E} \to \mathbb{R}^m$ with $\mathbb{E} \subseteq \mathbb{R}^n$. The linear map $Df_{\tilde{\boldsymbol{x}}} : \mathbb{E} \to \mathbb{R}^m$ is called the total derivative of f at $\tilde{\boldsymbol{x}}$. Conceptually, the definition of the total derivative expresses the idea that $Df_{\tilde{\boldsymbol{x}}}$ is the best linear approximation of f at point $\tilde{\boldsymbol{x}}$:*

$$f(\tilde{\boldsymbol{x}} + \boldsymbol{h}) = f(\tilde{\boldsymbol{x}}) + Df_{\tilde{\boldsymbol{x}}}(\boldsymbol{h}) + o(\|\boldsymbol{h}\|) \quad \forall\, \boldsymbol{h} \in \mathbb{E}\,, \;\; \tilde{\boldsymbol{x}} + \boldsymbol{h} \in \mathbb{E}$$

Instead of $Df_{\tilde{\boldsymbol{x}}}(h)$ we can also write $J_f(\tilde{\boldsymbol{x}})\,\boldsymbol{h}$, where $J_f(\tilde{\boldsymbol{x}})$ is called the Jacobian at $\tilde{\boldsymbol{x}}$.

Corollary 1 *Let $f : \mathbb{E} \subseteq \mathbb{R}^n \to \mathbb{R}^m$ with $Df_{\boldsymbol{x}} : \mathbb{E} \to \mathbb{R}^m$ the total derivative of f at $\boldsymbol{x} \in \mathbb{E}$. For an infinitesimal change $dx \in \mathbb{E}$, at the point $\boldsymbol{x}$, the corresponding infinitesimal change in the output, $df \in \mathbb{R}^m$, is given by:*

$$df = Df_{\boldsymbol{x}}\, dx$$

Theorem 1 *Properties of the total derivative as defined in Definition 1:*

i. Let $f : \mathbb{E} \to \mathbb{R}^m$ and $g : \mathbb{E} \to \mathbb{R}^m$, with $\mathbb{E} \subseteq \mathbb{R}^n$, be totally differentiable at $\tilde{\boldsymbol{x}}$. Then,

$$D(f+g)_{\tilde{\boldsymbol{x}}} = Df_{\tilde{\boldsymbol{x}}} + Dg_{\tilde{\boldsymbol{x}}}$$

ii. Let $a \in \mathbb{R}$ and $f : \mathbb{E} \to \mathbb{R}^m$ with $\mathbb{E} \subseteq \mathbb{R}^n$, be totally differentiable at $\tilde{\boldsymbol{x}}$. Then,

$$D(af)_{\tilde{\boldsymbol{x}}} = aDf_{\tilde{\boldsymbol{x}}}$$

iii. Let $f : \mathbb{E} \subseteq \mathbb{R}^n \to \mathbb{R}^m$ and $g : \mathbb{V} \subseteq \mathbb{R}^k \to \mathbb{R}^n$, with g totally differentiable at $\tilde{\boldsymbol{x}} \in \mathbb{R}^k$ and f totally differentiable at $g(\tilde{\boldsymbol{x}}) \in \mathbb{R}^n$. Then,

$$D(f \circ g)_{\tilde{\boldsymbol{x}}} = Df_{g(\tilde{\boldsymbol{x}})} Dg_{\tilde{\boldsymbol{x}}}$$

iv. (Differential Product Rule). Let $f : \mathbb{E} \to \mathbb{R}^{m \times p}$ and $g : \mathbb{E} \to \mathbb{R}^{p \times q}$, with $\mathbb{E} \subseteq \mathbb{R}^n$, be totally differentiable at $\boldsymbol{x} \in \mathbb{E}$. Then,

$$d(fg) = (df)\, g + f\, (dg)$$

Here, d represents the differential operator as in Corollary 1, capturing the infinitesimal change in the product of f and g in response to an infinitesimal change in their inputs at the point $\boldsymbol{x}$.

Bibliography

[ACT19] Francisco J. Aragón Artacho, Rubén Campoy, and Matthew K. Tam. *The Douglas-Rachford Algorithm for Convex and Nonconvex Feasibility Problems*. 2019. arXiv: 1904.09148 [math.OC]. URL: https://arxiv.org/abs/1904.09148.

[Bec17] Amir Beck. *First-Order Methods in Optimization*. Philadelphia, PA, USA: SIAM-Society for Industrial and Applied Mathematics, 2017. ISBN: 1611974984.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[Fig01] Mário Figueiredo. "Adaptive Sparseness Using Jeffreys Prior". In: *Advances in Neural Information Processing Systems*. Ed. by T. Dietterich, S. Becker, and Z. Ghahramani. Vol. 14. MIT Press, 2001. URL: https://proceedings.neurips.cc/paper_files/paper/2001/file/dd055f53a45702fe05e449c30ac80df9-Paper.pdf.

[GBC16] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[Str21] Gilbert Strang. *Introduction to Linear Algebra*. 5th ed. Cambridge University Press, 2021. ISBN: 9781733146654.

[WC13] Weiran Wang and Miguel Á. Carreira-Perpiñán. *Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application*. 2013. arXiv: 1309.1541 [cs.LG]. URL: https://arxiv.org/abs/1309.1541.