EE364b Spring 2023 Homework 3

Due Friday 4/30 at 11:59pm via Gradescope

3.1 (4 points) *Constrained subgradient method.* Consider the optimization problem

$$\begin{array}{ll} \text{minimize}_{\{x_j\}_{j=1}^J} & f(x_1,\ldots,x_J) := \|b - \sum_{j=1}^J A_j x_j\|_1 \\ & \text{s.t. } A_j x_j \geq 0 \text{ and } \|x_j\|_2 \leq \lambda, \forall j \in \{1,2,\ldots,J\}, \end{array}$$

with variable $x_1,\ldots,x_J \in \mathbf{R}^n$ and problem data $A_1,\ldots,A_J \in \mathbf{R}^{m\times n}$, $b \in \mathbf{R}^m$ and $\lambda > 0$. Since the ℓ_1 loss puts less emphasis on poorly-fit outliers than the typical least-squares loss, this can be interpreted as a *robust regression* problem. We will apply the subgradient method for constrained optimization given on page 11 of the lecture slides.

(a) Let $J = 3$, $n = 100$, $m = 10$, and $\lambda = 1$. Generate random matrices $A_1,\ldots,A_J \in \mathbf{R}^{m\times n}$ with independent uniformly distributed entries in the interval $\left[0, \frac{1}{\sqrt{m}}\right)$ and, random vectors $x_1,\ldots,x_J \in \mathbf{R}^n$ with independent uniformly distributed entries in the interval $\left[0, \frac{1}{\sqrt{n}}\right)$, then set $b = \sum_{j=1}^J A_j x_j$. Initialize the constrained subgradient methods by sampling each block as $x_i^0 \sim \mathcal{N}(0, I/\sqrt{n})$. Plot convergence in terms of the objective $f(x_1^{(k)},\ldots,x_J^{(k)})$. Try different step length schedules and compare them in the same plot. Which works best? Also, plot the maximal violation of the constraints at each step.

(b) Implement the primal-dual subgradient method and compare with the constrained subgradient method using the problem data you generated in Part (a). Which method is faster?

3.2 (5 points) *Stochastic Log-optimal portfolio optimization using return oracle.* We consider the portfolio optimization problem

$$\begin{array}{ll} \text{maximize} & \mathbf{E}_r \log(r^T x) - \frac{\lambda}{2}\|x\|_2^2 \\ \text{subject to} & \mathbf{1}^T x = 1, \qquad x \succeq 0, \end{array}$$

with variable (portfolio weights) $x \in \mathbf{R}^n$ and ℓ^2-regularization parameter $\lambda > 0$. The expectation is over the distribution of the (total) return vector $r \in \mathbf{R}^n_{++}$, which is a random variable. The ℓ^2-regularization parameter λ lets us trade off between maximizing the returns of the portfolio and evenly distributing the portfolio weights over different investments. (Although not relevant in this problem, the log-optimal portfolio maximizes the long-term growth of an initial investment, assuming the investments are re-balanced to the log-optimal portfolio after each investment period, and ignoring transaction costs.)

In this problem we assume that we do not know the distribution of r (other than that we have $r \succ 0$ almost surely). However, we have access to an oracle that will generate independent samples from the return distribution. (Although not relevant, these samples could come from historical data, or stochastic simulations, or a known or assumed distribution.)

(a) (1 point) Explain how to use the (projected) stochastic subgradient method, using one return sample for each iteration, to find (in the limit) a log-optimal portfolio. Describe how to carry out the projection required, and how to update the portfolio in each iteration.

Hint. Note that projecting $x^{(k)}$ onto the constraints involves projecting onto a simplex, or solving the optimization problem

$$\begin{array}{ll} \text{minimize} & (1/2)\|z - x^{(k)}\|_2^2 \\ \text{subject to} & \mathbf{1}^T z = 1, \quad z \succeq 0, \end{array} \tag{1}$$

with variable z. One way to solve the problem is to introduce a dual variable ν for the equality constraint and write the (partial) Lagrangian as

$$L(z, \nu) = (1/2)\|z - x^{(k)}\|_2^2 + \nu(\mathbf{1}^T z - 1),$$

with $\mathbf{dom}\, L(z, \nu) = \mathbf{R}^n_+ \times \mathbf{R}$ (in other words, the inequality constraint $z \succeq 0$ is implicit). Consider the single variable function $g(\nu)$ obtained by minimizing $L(z, \nu)$ over z. The value $\nu^\star$ that maximizes $g(\nu)$ can be found using bisection, and the solution $z^\star$ to problem (1) can be found given $\nu^\star$.

(b) (1 point) Implement the method and run it on the problem with $n = 10$ assets, with return sample oracle `log_opt_return_sample` in the files `hw3_utils.py`, `hw3_utils.jl`, and `log_opt_return_sample.m`. This function called with argument m returns an $n \times m$ matrix whose columns are independent return samples. We have also provided Python/Julia/MATLAB functions to project onto the simplex (see Canvas/Files/Homeworks).

You are welcome to look inside `hw3_utils.py` to see how we are generating the sample. The distribution is a mixture of two log-normal distributions; you can think of one as the standard return model and the other as the return model in some abnormal regime. However, your stochastic subgradient algorithm can only call `log_opt_return_sample(1)`, once per iteration; you cannot use any information found inside the file in your implementation.

To get a Monte Carlo approximation of the objective function value, you can generate a block of, say, 10^5 samples (using `R_emp=log_opt_return_sample(1e5)`, which only needs to be done once). In Python, you can then use `obj_hat = np.mean(np.log(R_emp @ x)) - lambd/2 * np.dot(x, x)` as your estimate of the objective function.

Run the algorithm with both $\lambda = 1$ and $\lambda = 10$. Plot the (approximate) objective value versus iteration, as well as the best approximate objective value obtained up to that iteration, and compare the solutions found for $\lambda = 1$ and $\lambda = 10$. (Note that evaluating the objective will require far more computation than each stochastic subgradient step.)

You may need to play around with the step size selection in your method to get reasonable convergence. Remember that your objective value evaluation is only an approximation.

(c) (1 point) Repeat part (b) without the constraint $x \geq 0$. In other words, solve the optimization problem

$$\begin{array}{ll} \text{maximize} & \mathbf{E}_r \log(r^T x) - \frac{\lambda}{2}\|x\|_2^2 \\ \text{subject to} & \mathbf{1}^T x = 1, \end{array}$$

using the same algorithm as in part (b) (though the projection onto the constraints will be different, because the constraints are different). As in part (b), run the algorithm for both $\lambda = 1$ and $\lambda = 10$, plot the (approximate) objective value versus iteration, as well as the best approximate objective value obtained up to that iteration, and compare the solutions for the different values of λ.

(d) (extra credit: 2 points) Repeat part (b) using Mirror Descent and stochastic subgradients.

3.3 *(4 points)* Consider the Bregman divergence $D(x, y) = \sum_i x_i \log(x_i/y_i) - (x_i - y_i)$, which is known as the generalized KL divergence.

Show that the projection with respect to this Bregman divergence on the simplex, *i.e.*, $\Delta_n = \{x \in \mathbf{R}^n_+ \mid \mathbf{1}^T x = 1\}$, amounts to a simple renormalization $y \to y/\|y\|_1$.

3.4 *(4 points) High dimensional problems, mirror descent, and gradient descent.* We consider using mirror descent versus projected subgradient descent to solve the non-smooth minimization problem

$$\text{minimize } f(x) = \max_{i \in \{1,\ldots,m\}} \{a_i^T x + b_i\} \;\; \text{subject to } x \in \Delta_n = \{z \in \mathbf{R}^n_+ \mid z^T \mathbf{1} = 1\}.$$

Implement mirror descent with the choice $h(x) = \sum_{i=1}^n x_i \log x_i$ and projected subgradient descent for this problem. (You will need to project onto the simplex efficiently for this to be a reasonable method at all.) You will compare the performance of these two methods.

Generate random problem data for the above objective with a_i drawn as i.i.d. $N(0, I_{n\times n})$ (multivariate normals) and b_i drawn i.i.d. $N(0, 1)$, where $n = 500$ and $m = 50$. Solve the problem using CVX (or `Convex.jl` or CVXPY), then run mirror descent and projected gradient descent on the same data for 100 iterations. Run each method with constant stepsizes $\alpha \in \{2^{-12}, 2^{-11}, \ldots, 2^6, 2^7\}$. Repeat this 25 times, then plot the average optimality gap $f(x^k) - f(x^\star)$ or $f_{\text{best}}^k - f(x^\star)$ as a function of iteration for the best stepsize (chosen by smallest optimality gaps) for each method. Which method gives the best performance?