EE364b Homework 2

Due Friday 4/19 at 11:59pm.

2.2 *A variation on alternating projections.* We consider the problem of finding a point in the intersection $\mathcal{C} \neq \emptyset$ of convex sets $\mathcal{C}_1, \ldots, \mathcal{C}_m \subseteq \mathbf{R}^n$. To do this, we use alternating projections to find a point in the intersection of the two sets

$$\mathcal{C}_1 \times \cdots \times \mathcal{C}_m \subseteq \mathbf{R}^{mn}$$

and

$$\{(z_1, \ldots, z_m) \in \mathbf{R}^{mn} \mid z_1 = \cdots = z_m\} \subseteq \mathbf{R}^{mn}.$$

Show that alternating projections on these two sets is equivalent to the following iteration: project the current point in $\mathbf{R}^n$ onto each convex set (independently in parallel), and then average the results. Draw a simple picture to illustrate this.

2.4 *Asynchronous alternating projections.* We consider the problem of finding a sequence of points that approach the intersection $\mathcal{C} \neq \emptyset$ of some convex sets $\mathcal{C}_1, \ldots, \mathcal{C}_m$. In the alternating projections method described in lecture, the current point is projected onto the farthest set. Now consider an algorithm where, at every step, the current point is projected onto any set not containing the point.

Give a simple example showing that such an algorithm can fail, *i.e.*, we can have $\mathbf{dist}(x^{(k)}, \mathcal{C}) \not\to 0$ as $k \to \infty$.

2.5 *Alternating projections to solve linear equations.* We can solve a set of linear equations expressed as

$$A_i x = b_i, \quad i = 1, \ldots, m,$$

where $A_i \in \mathbf{R}^{m_i \times n}$ are fat and full rank, using alternating projections on the sets

$$\mathcal{C}_i = \{z \mid A_i z = b_i\}, \quad i = 1, \ldots, m.$$

Assuming that the set of equations has solution set $\mathcal{C} \neq \emptyset$, we have $\mathbf{dist}(x^{(k)}, \mathcal{C}) \to 0$ as $k \to \infty$.

We take the approach of projecting the current point onto all the sets, and forming our next iterate as the average of these projected points, as in exercise (2.2).

(a) Consider the case $m_i = 1$ and $m = n$ (*i.e.*, we have n scalar equations), and assume there is a unique solution $x^\star$ of the equations. Work out an explicit formula for the update, and show that the error $x^{(k)} - x^\star$ satisfies a (time-varying) linear dynamical system.

(b) Generate random Gaussian matrix $A \in \mathbf{R}^{m \times n}$ and vector $x \in \mathbf{R}^n$. Set $b = Ax$. Apply the iterative method derived above and plot convergence in terms of the distance of the iterate to the solution x and in terms of the residual $\|Ax^{(k)} - b\|_2$.

(c) Apply the iterative method to your example from exercise (2.4) and plot convergence. Show the results as well when applying the asynchronous alternating projection algorithm from exercise (2.4).

2.6 *L1 regression* Consider the optimization problem

$$\text{minimize} \quad \|Ax - b\|_1$$

with variable $x \in \mathbf{R}^n$ and problem data $A \in \mathbf{R}^{m \times n}$ and $b \in \mathbf{R}^m$. We will apply the subgradient method.

(a) Show that the subgradient method with Polyak's step length updates the current point to a point at which the first order (linear) approximation has value f^* (optimal value).

(b) As in exercise (2.5), generate random Gaussian matrix $A \in \mathbf{R}^{m \times n}$ and vector $x \in \mathbf{R}^n$, then set $b = Ax$. Plot convergence in terms of distance to the solution x and in terms of the objective $\|Ax^{(k)} - b\|_1$. Try different step length schedules, including Polyak's step length.

2.9 *Subgradient method for total variation in-painting.* A grayscale image is represented as an $m \times n$ matrix of intensities U^{orig} (typically between the values 0 and 255). You are given the values U^{orig}_{ij}, for $(i,j) \in \mathcal{K}$, where $\mathcal{K} \subset \{1, \ldots, m\} \times \{1, \ldots, n\}$ is the set of indices corresponding to known pixel values. Your job is to *in-paint* the image by guessing the missing pixel values, *i.e.*, those with indices not in $\mathcal{K}$. The reconstructed image will be represented by $U \in \mathbf{R}^{m \times n}$, where U matches the known pixels, *i.e.*, $U_{ij} = U^{\text{orig}}_{ij}$ for $(i,j) \in \mathcal{K}$.

The reconstruction U is found by minimizing the total variation of U, subject to matching the known pixel values. We will use the ℓ_2 total variation, defined as

$$\mathbf{tv}(U) = \sum_{i=1}^{m-1} \sum_{j=1}^{n-1} \left\| \begin{bmatrix} U_{i+1,j} - U_{i,j} \\ U_{i,j+1} - U_{i,j} \end{bmatrix} \right\|_2 .$$

Note that the norm of the discretized gradient is *not* squared.

(a) Explain how to find a subgradient $G \in \partial\, \mathbf{tv}(U)$. It is sufficient to give a formula for G_{ij}.

(b) Implement a projected subgradient method for minimizing $\mathbf{tv}(U)$ subject to $U_{ij} = U^{\text{orig}}_{ij}$ for $(i,j) \in \mathcal{K}$.

Use it to solve the problem instance given in `subgrad_tv_inpaint_data.m`. You will also need `tv_l2_subgrad.m`, `lena512.bmp`, and `lena512_corrupted.bmp`.

Show the original image, the corrupted image, and the in-painted image. Plot $\textbf{tv}\left(U^{(k)}\right)$ ($U^{(k)}$ is U in the kth iteration) versus k.

The file `subgrad_tv_inpaint_data.m` defines `m`, `n`, and matrices `Uorig`, `Ucorrupt`, and `Known`. The matrix `Ucorrupt` is `Uorig` with the unknown pixels whited out. The matrix `Known` is $m \times n$, with (i, j) entry one if $(i, j) \in \mathcal{K}$ and zero otherwise. The file also includes code to display `Uorig` and `Ucorrupt` as images.

Writing matlab code that operates quickly on large image matrices is tricky, so we have provided a function `tv_l2_subgrad.m` that computes $\textbf{tv}(U)$ and $G \in \partial\,\textbf{tv}(U)$ given U. `tv_l2_subgrad.m` uses the `norms` function from `CVX`, so you will need `CVX` installed. A simple (and fast) way to set the known entries of a matrix `U` to their known values is `U(Known == 1) = Uorig(Known == 1)`.

You may need to try several step length sequences to get fast enough convergence. We obtained good results with step sizes like $\alpha_k = 1000/k$ and $\alpha_k = 50/\sqrt{k}$, but feel free to experiment with others. Do not hesitate to run the algorithm for 1000 or more iterations.

Once it's working, you might like to create an animated GIF that shows algorithm progress, say, displaying U every 50 iterations. We used the function `imwrite(U_record,'inpaint.gif','DelayTime',1,'LoopCount',inf)`. Here `U_record` is an $m \times n \times 1 \times r$ matrix, where `U_record(:, :, 1, i)` is the ith intermediate value of U out of the r stored in `U_record`. `imwrite` will project invalid intensity values into the range $[0, 255]$ (with a warning).