# CSCI 6962/4140: Homework 2

Assigned Thursday, September 26, 2024. Due by 11:59pm Thursday October 10, 2024.

Create a Jupyter notebook for this assignment, and use Python 3. Write documented, readable and clear code (e.g. use reasonable variable names). Submit this notebook interspersing any textual answers in Markdown cells (using LaTeX), clearly labeled, along with your code.

This homework will build up to the proximal gradient descent algorithm, then show its use in solving image deblurring problems.

1. [5 pts] The proximal operator of a convex function $g$ (that may be nonsmooth) is defined as the solution to the optimization problem

$$\operatorname{prox}_g(\mathbf{v}) = \operatorname{argmin}_{\mathbf{x}}\ g(\mathbf{x}) + \frac{1}{2}\|\mathbf{x}-\mathbf{v}\|_2^2. \tag{Prox}$$

Consequently, $\operatorname{prox}_g(\mathbf{v})$ is a point that balances between minimizing $g$ and staying close to $\mathbf{v}$.

- Explain concisely why (Prox) has a unique solution.
- What optimality condition must be satisfied for $\mathbf{x}^\star$ to be the minimizer of (Prox)?
- Let $\alpha$ be a nonnegative real number. Predict the behavior of $\operatorname{prox}_{\alpha g}(\mathbf{v})$ as $\alpha \to 0$, and as $\alpha \to \infty$.

When $\operatorname{prox}_g$ has a closed form or (Prox) can be solved efficiently, we say that $g$ is prox-friendly.

2. [5 pts] The soft-shrinkage operator on vectors is defined by

$$S_\alpha(\mathbf{x}_0) = \operatorname{prox}_{\alpha\|\cdot\|_1}(\mathbf{x}_0) = \operatorname{argmin}_{\mathbf{x}\in\mathbb{R}^d}\ \|\mathbf{x}\|_1 + \frac{1}{2\alpha}\|\mathbf{x}-\mathbf{x}_0\|_2^2$$

for any $\alpha > 0$. Argue that

$$S_\alpha(\mathbf{x}_0)_i = s_\alpha((\mathbf{x}_0)_i),$$

where $s_\alpha$ is the scalar soft-shrinkage operator from Participation 5. Thus the $\ell_1$ norm is prox-friendly.

Write a function `softShrink(x0, alpha)` that implements the soft-shrinkage operator (efficiently, so no for loops).

3. [10 pts] The proximal gradient descent algorithm[1] is used to solve convex optimization problems of the form

$$\min_{\mathbf{x}} f(\mathbf{x}) + g(\mathbf{x}) \tag{Comp}$$

where $f$ is a differentiable convex function and $g$ is a (prox-friendly) nondifferentiable convex function. This is called a composite optimization problem because it requires minimizing the sum of two functions. The prox-gradient algorithm takes advantage of structures in $f$ (differentiablity) and $g$ (prox-friendliness) to solve this convex composite optimization problem.

To move from iterate $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$, the method first takes a gradient descent step of $f$, then chooses $\mathbf{x}_{t+1}$ to balance between minimizing $g(\mathbf{x})$ and staying close to the intermediate iterate:

$$\begin{aligned}\mathbf{x}_{t+1} &= \operatorname{prox}_{\alpha_t g}(\mathbf{x}_t - \alpha_t\nabla f(\mathbf{x}_t)) \qquad \text{(ProxGrad)}\\ &= \operatorname{argmin}_{\mathbf{x}}\ \frac{1}{2\alpha_t}\|\mathbf{x}-(\mathbf{x}_t-\alpha_t\nabla f(\mathbf{x}_t))\|_2^2 + g(\mathbf{x}).\end{aligned}$$

- Justify the use of the proximal gradient algorithm by arguing that if $\mathbf{x}^\star$ is a fixed point of (ProxGrad) then it is a minimizer of (Comp).

[1] Also known as composite gradient descent, generalized gradient descent, the prox-linear algorithm, or forward-backward splitting : this is a *really* popular and ubiquitous algorithm.

- Simplify the expression for $\mathbf{x}_{t+1}$ when $g = 0$. What is the name of the resulting algorithm?

The proximal gradient algorithm is frequently employed to solve optimization problems involving nonsmooth regularizers. A canonical example of this use case is the ISTA algorithm below for the LASSO problem.

4. [10 pts] Consider the LASSO problem

$$\mathbf{x}^\star = \text{argmin}_{\mathbf{x}\in\mathbb{R}^d} \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{x}\|_1. \tag{LASSO}$$

   Write the expression for $\mathbf{x}_{t+1}$ in the composite gradient descent method for solving (LASSO), using the soft-shrinkage operator. Assume that stepsizes $\alpha_t$ are given.

   Write a function `[xT, objhist] = istaLasso(A, b, lambda_reg, x0, alpha, T)` that returns $x_T$, the $T$th iterate of the composite gradient method for (LASSO) when the initial iterate is `x0` and the stepsize constant is `alpha`; it should also return a vector containing the objective values at each of the iterates $\mathbf{x}_0, \ldots, \mathbf{x}_T$. Use your function `softShrink`.

5. [10 pts] Compute the subdifferential of the objective of (LASSO) at a point $\mathbf{x}$.

   Write a function `g = lassoSubgrad(A, b, lambda_reg, x)` that returns a subgradient for the LASSO objective at `x`.

   Write a function `[xT, objhist] = subgradLasso(A, b, lambda_reg, x0, alpha, T)` that returns the last iterate of the subgradient descent method for solving (LASSO) when the initial iterate is `x0` and the stepsize is constant and given by `alpha`; it should also return a vector containing the value of the objective function at iterates $\mathbf{x}_0, \ldots, \mathbf{x}_T$. Use your function `lassoSubgrad`.

6. [60 pts] Use the ISTA and subgradient solvers for the LASSO problem to solve the deblurring problem. The setup here is that we assume we observe a blurred and noisy version of an image, $\mathbf{b} = \mathbf{B}\mathbf{x} + \mathbf{e}$, and given knowledge of $\mathbf{B}$, our task is to recover an approximation of $\mathbf{x}$. Note that we are representing images as vectors by stacking the columns of the image together into a vector representation. Thus an image of size $n \times n$ is represented as a vector in $\mathbb{R}^{n^2}$.

   In general the blur matrix $\mathbf{B}$ is *not invertible*, but we could solve a least-squares problem to obtain an estimate of the image,

$$\mathbf{x}_{\text{LS}} = \text{argmin}_{\mathbf{w}}\|\mathbf{B}\mathbf{w} - \mathbf{b}\|_2.$$

   This method doesn't work well if $\mathbf{B}$ is not close to invertible or in the presence of noise. In these cases, we want to take advantage of additional expectations on the image $\mathbf{x}$ to design a convex optimization problem whose solution is a better estimator of $\mathbf{x}$.

   In this exercise, we will use the popular modeling assumption that *natural images are approximately sparse in an appropriately chosen basis.* We will assume that there is a known orthonormal basis $\mathbf{H}$ in which $\mathbf{x}$ is approximately sparse: i.e., there is a vector $\mathbf{z}$ for which $\mathbf{x} = \mathbf{H}\mathbf{z}$, and $\|\mathbf{z}\|_1$ is small. This implies that $\mathbf{b} = \mathbf{B}\mathbf{H}\mathbf{z} + \mathbf{e} = \mathbf{A}\mathbf{z} + \mathbf{e}$, where $\mathbf{A} = \mathbf{B}\mathbf{H}$. Then we solve the LASSO problem

$$\mathbf{z}^\star = \text{argmin}_{\mathbf{z}} \frac{1}{2}\|\mathbf{A}\mathbf{z} - \mathbf{b}\|_2^2 + \lambda\|\mathbf{z}\|_1$$

   for a judicious choice of $\lambda$ and form the estimate $\mathbf{x}_{\text{LASSO}} = \mathbf{H}\mathbf{z}^\star$.

   A standard choice of $\mathbf{H}$ is given by the Haar wavelet basis, and for the data set and noise level you will use, I have found $\lambda = 2 \times 10^{-4}$ to give good results.

- Download utility code for working with Haar matrices, blurring matrices, and converting between vectors and matrices from `https://www.cs.rpi.edu/~gittea/old-site/teaching/fall2023/files/deblurringUtils.py`.
- Load the image we will be working with, resize it[2] so the problem is tractable, convert it to float format, and take a look at the image.

```
from deblurringUtils import *
from skimage import data

n = 128
camera = data.camera()
smallcamera = resize(camera, [n, n])
camdata = img_as_float(smallcamera)
visualize(camdata)
```

- Verify that the Haar basis does allow an almost sparse representation of the image

```
H = vectorized2DHaarMatrix(n, n) # the Haar matrix, an orthonormal basis
haarcoeffs = H.T @ vectorize(camdata) # convert to the Haar basis

p = 90 # what percentage of the coefficients to set to zero
thres = np.percentile(np.abs(haarcoeffs), p)
haarcoeffs[abs(haarcoeffs) <= thres] = 0  # set most entries to zero

# reconstruct the image from the sparse representation
reconstim = unvectorize(H @ haarcoeffs, n, n)
visualize(reconstim)
```

- Verify that a similarly sparse representation in the original pixel space is much worse.

```
import numpy.random as random

# mask the same percentage of pixels, randomly
mask = np.ones(n**2)
mask[:int(p/100*n**2)] = 0
mask = unvectorize(random.permutation(mask), n, n)
visualize(camdata * mask )
```

- Create the blurred, corrupted image. Corrupt using i.i.d. $\mathcal{N}(0, 1 \times 10^{-4})$ Gausian noise.

```
B = vectorized2DBlurMatrix(n, n, 5);
std = 1e-2
corruption = std * random.randn(n**2)
b = B @ vectorize(smallcamera) + corruption
blurredcam = unvectorize(b, n, n)
visualize(blurredcam)
```

- Compute and visualize the naive solution $\mathbf{x}_{\text{LS}}$.

```
from scipy.sparse.linalg import gmres
from scipy.sparse import csr_matrix

# solving with sparse matrices is faster
sB = csr_matrix(B)
linres, _ = gmres(sB, b, maxiter=50)
visualize(unvectorize(linres, n, n))
```

[2]You may use a smaller number during debugging, but $n$ must be a power of two.

- Define the matrix $\mathbf{A}$ to be used in the LASSO problem, and compute the $\beta$-smoothness parameter of the smooth part of the LASSO objective, $\beta = \|\mathbf{A}^T\mathbf{A}\|_2$. Set the constant stepsizes to $\alpha = \frac{1}{\beta}$ for both methods.

  ```
  from scipy.sparse.linalg import svds

  A = B @ H
  _, topsv, _ = svds(A, k=1)
  alphaSubgrad = 1/topsv**2
  ```

  What is the stepsize for the subgradient method?
- Set $\lambda = 2 \times 10^{-4}$ as the regularization parameter and use 1000 steps of the subgradient solver to recover the image. Visualize the recovered image. What is the final objective value?
- Using the same value of $\lambda$, use 1000 steps of ISTA to recover the image. Visualize the recovered image. What is the final objective value?
- Plot the objective values of ISTA and the subgradient solvers on a log-scale. What conclusions do you draw about the relative merits of the two approaches to solving the LASSO problem? What are the advantages and disadvantages of solving the LASSO problem vs the least squares problem?
- We plotted the LASSO objective to verify visually that the methods seem to be converging. Comment on the meaningfulness (or not) of that value in measuring the *quality* of the recovered solution.

7. (CSCI 6961 students) [30 pts] Surprisingly, using large enough finite values of the regularization parameter $\lambda$ ensures that the solution to (LASSO) is exactly 0. In fact, we can prove something much stronger!

   (a) Assume $f$ is a smooth convex function and we know *a priori* that $\|\nabla f(\mathbf{x}^\star)\|_2 < C$ for some constant $C$, where $\mathbf{x}^\star$ is the solution to

   $$\operatorname{argmin}_{\mathbf{x}}\ f(\mathbf{x}) + \lambda\|\mathbf{x}\|_1.$$

   Use Fermat's condition to give a bound on the number of nonzero entries in $\mathbf{x}$ in terms of $C$ and $\lambda$. Thus we see that by increasing $\lambda$ when using $\ell_1$ regularization, we can ensure that $\mathbf{x}^\star$ is increasingly sparse.

   The following steps find such a guarantee on the sparsity of the solution to the LASSO problem.

   (b) Establish that for any $d$-dimensional vector, $\|\mathbf{x}\|_2 \leq \|\mathbf{x}\|_1$.

   (c) Argue that the solution to (LASSO) satisfies $\|\mathbf{x}^\star\|_1 \leq \|\mathbf{x}_{\text{OLS}}\|_1$, where $\mathbf{x}_{\text{OLS}}$ is the solution to the OLS problem, and show that it follows that

   $$\|\mathbf{A}^T(\mathbf{A}\mathbf{x}^\star - \mathbf{b})\|_2 \leq \|\mathbf{A}^T\mathbf{A}\|_2\|\mathbf{x}_{\text{OLS}}\|_1 + \|\mathbf{A}^T\mathbf{b}\|_2.$$

   (d) Use this bound to give a constant $\lambda_0$ that ensures the solution to (LASSO) is 0 when $\lambda > \lambda_0$.