Assignment 4

Numerical Optimization WS 2022

Lea Bogensperger, lea.bogensperger@icg.tugraz.at
Nives Krizanec, nives.krizanec@student.tugraz.at
Jakob Ederer, ederer@student.tugraz.at

January 17, 2023

Deadline: Feb 7th, 2023 at 23:59

Submission: Upload your report and your implementation to the TeachCenter. Please use the provided framework-file for your implementation. Make sure that the total size of your submission does not exceed 50MB. Include **all** of the following files in your submission:

- `report.pdf`: This file includes your notes for the individual tasks. Keep in mind that we must be able to follow your calculation process. Thus, it is not sufficient to only present the final results. You are allowed to submit hand written notes, however a compiled LaTeX document is preferred. In the first case, please ensure that your notes are well readable.
- `main.py`: This file includes your python code to solve the different tasks. Please only change the marked code sections. Also please follow the instructions defined in `main.py`.
- `figures.pdf`: This file is generated by running `main.py`. It includes a plot of all mandatory figures on separate pdf pages. Hence, you do not have to embed the plots in your report.

Optimization over Convex Sets

In this exercise your task is to investigate algorithms to solve a class of constrained optimization problems, i.e.

$$\min_{\mathbf{x}} f(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{x} \in C \ , \tag{1}$$

where $f(\mathbf{x})$ continuously differentiable and C convex. In particular we will have a closer look at the *Frank-Wolfe* algorithm. While the projected gradient descent algorithm extends the standard steepest descent algorithm by introducing a projection step which assures that $\mathbf{x}_k \in C$ for $k = 1, ..., K$. While the difficult aspect of the projected gradient descent algorithm is finding the projection operator, the Frank-Wolfe (FW) algorithm provides a different approach. To solve a problem of the general form stated in eq. (1), the FW algorithm in each iteration aims to find a step direction

$$\mathbf{p}_k = \arg\min_{\mathbf{p}} \nabla f(\mathbf{x}_k)^T \mathbf{p} \quad \text{s.t.} \quad \mathbf{p} \in C \ . \tag{2}$$

The newly attained $\mathbf{p}_k$ is then in turn used to perform an update step based on a convex combination with the previous point $\mathbf{x}_k$. The full FW algorithm is stated in algorithm 1.

Algorithm 1: Frank-Wolfe (Conditional Gradient) algorithm

Choose $x_1 \in C$
for $k > 0$ **do**
  $\mathbf{p}_k = \arg\min_{\mathbf{p}} \nabla f(\mathbf{x}_k)^T \mathbf{p} \quad \text{s.t.} \quad \mathbf{p} \in C$.
  $\tau_k = \frac{2}{k+1}$
  $\mathbf{x}_{k+1} = (1 - \tau)\mathbf{x}_k + \tau_k \mathbf{p}_k$
end

1 Signal Denoising (15 P.)

In this first exercise your task is to denoise a given signal $\mathbf{b} \in \mathbb{R}^n$ with superimposed Gaussian noise. The relation between the clean signal $\hat{\mathbf{b}}$ and the noisy signal is thus given by

$$b_i = \hat{b}_i + \eta \quad \text{with} \quad \eta \sim \mathcal{N}(0, \sigma^2), \quad i = 1, ..., n \ . \tag{3}$$

To bring our problem into the format given in eq. (1) we define

$$\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2 \quad \text{s.t.} \quad \mathbf{x} \in \Delta = \left\{ \mathbf{x} \in \mathbb{R}^d \mid x_j \geq 0, \sum_{j=1}^{d} x_j = 1 \right\} \ , \tag{4}$$

where the constraint set Δ is called unit simplex. The matrix $\mathbf{A}$ represents a dictionary where each column (or dictionary entry) corresponds to one of d basis functions of the *discrete cosine transform* (DCT). By solving the optimization problem in eq. (4) we thus find a weighting vector $\mathbf{x}$ of DCT basis functions that yield our smooth result signal. Each basis function/vector $\mathbf{a}_j$ with $j = 1, ..., d$ can be computed through

$$(\mathbf{a}_j)_i = \alpha_j \cos\left(\frac{\pi}{n}(j-1)\left(i - \frac{1}{2}\right)\right) \quad \text{with} \quad \alpha_j = \begin{cases} \frac{1}{\sqrt{n}} & \text{if } j = 1 \\ \sqrt{\frac{2}{n}} & \text{else} \end{cases} \ , \tag{5}$$

where $(\mathbf{a}_j)_i$ denotes the i-th element of basis vector $\mathbf{a}_j$.

Tasks:

With Pen & Paper:

1. Analytically determine the optimal solution to find $\mathbf{p}_k$ for the FW algorithm given the unit simplex constraint.

In Python:

3. Construct the dictionary matrix $\mathbf{A}$ by computing its basis functions $\mathbf{a}_j$ according to eq. (5).
4. Implement the Frank-Wolfe algorithm.
5. Create 4 individual experiments where
   (a) $d = 15$ and $\sigma^2 = 0.01^2$
   (b) $d = 15$ and $\sigma^2 = 0.03^2$
   (c) $d = 100$ and $\sigma^2 = 0.01^2$
   (d) $d = 5$ and $\sigma^2 = 0.01^2$

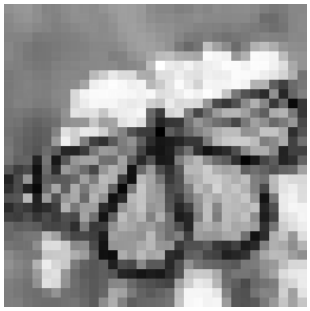
   For each experiment add Gaussian noise to the clean signal according to eq. (3). Then run the FW algorithm. Choose K such that the algorithm converges. Create a plot for each experiment showing the clean signal, the noisy signal and the results from using the Frank-Wolfe algorithm.
6. Discuss each experiment and state how the chosen parameter for d and σ^2 affect the outcome.

2 Image Compression (10 P.)

We now elaborate the use-cases of the FW-algorithm to a multi-dimensional case. Instead of a 1D signal we now deal with a 2D image as our input of size $n \times n$ using $n = 256$. The task is to compress the image as it is done in classic image compression algorithms, again using DCT basis functions. Since our image is rather large in dimensions the compression is done on a per-patch level. This means that we decompose our image into non-overlapping blocks of size $n_B \times n_B$ (we use $n_B = 8$ here) for each of which the compression is applied. Our constrained optimization now has the following form

$$\min_{\mathbf{x}} \sum_{s=1}^{B} \frac{1}{2}\|\mathbf{A}\mathbf{x}_s - \mathbf{b}_s\|_2^2 \quad \text{s.t.} \quad \mathbf{x}_s \in C = \left\{ \mathbf{x}_s \in \mathbb{R}^{n_B^2} \mid \|\mathbf{x}_s\|_1 \leq t \right\}, \ t > 0, \tag{6}$$

(a) Input image



(b) Compressed image

Figure 1: 256×256 image compression of 8×8 non-overlapping patches.

where the scaled ℓ_1 norm as a constraint ensures the compression and $B \in \mathbb{N}$ denotes the number of non-overlapping blocks that make up the image.

Each DCT basis image $\mathbf{a}_{lm}$ can be computed in analogy to eq. (5) by

$$(\mathbf{a}_{lm})_{ij} = \alpha_l \alpha_m \cos\left(\frac{\pi}{n}(l-1)\left(i-\frac{1}{2}\right)\right)\cos\left(\frac{\pi}{n}(m-1)\left(j-\frac{1}{2}\right)\right) \tag{7}$$

$$\text{with} \quad \alpha_l = \begin{cases} \frac{1}{\sqrt{n}} & \text{if } l=1 \\ \sqrt{\frac{2}{n}} & \text{else} \end{cases}, \quad \alpha_m = \begin{cases} \frac{1}{\sqrt{n}} & \text{if } m=1 \\ \sqrt{\frac{2}{n}} & \text{else} \end{cases}, \quad i,j = 1,...,n\ , \tag{8}$$

where the indices $l = 1, \cdots, d$ and $m = 1, \cdots, d$ denote the index of the image and i, j denote the index of the pixel. With this the dictionary can be constructed according to

$$\mathbf{A} = \left[\bar{\mathbf{a}}_{11}, \quad \bar{\mathbf{a}}_{12}, \quad ..., \quad \bar{\mathbf{a}}_{1d}, \quad \bar{\mathbf{a}}_{21}, \quad ..., \quad \bar{\mathbf{a}}_{2d}, \quad ..., \bar{\mathbf{a}}_{dd}\right]\ ,$$

with $\bar{\mathbf{a}}_{lm}$ denoting the flattened dictionary image.

Tasks:

Given an input image of size 256×256:

With Pen & Paper:

1. Determine the number of non-overlapping blocks B when using $n_B = 8$. Bear in mind that we have to set $i, j = 1, \cdots, n_B$ in (8).
2. Analytically determine the optimal solution to find $\mathbf{p}_k$ for the FW algorithm given the scaled ℓ_1 constraint.
3. For image compression using a DCT dictionary (which can be written as an orthogonal matrix, i.e. $A^{-1} = A^T$), there is actually an easier closed-form solution to the regularized, unconstrained optimization problem

$$\min_{\mathbf{x}} \sum_{s=1}^{B} \frac{1}{2}\|\mathbf{A}\mathbf{x}_s - \mathbf{b}_s\|_2^2 + \lambda\|\mathbf{x}_s\|_1, \tag{9}$$

where $\lambda \in \mathbb{R}^+$ is related to the amount of compression. This is also known as the LASSO model (least absolute shrinkage and selection operator). The closed-form solution then reads as

$$\mathbf{x}_s = \max(0, |\mathbf{A}^T\mathbf{b}_s| - \lambda)\ \text{sgn}(\mathbf{A}^T\mathbf{b}_s), \tag{10}$$

where $\text{sgn}(\cdot)$ is the sign function which is applied elementwise and it is defined as

$$\text{sgn(r)} = \begin{cases} +1 & \text{if } r \geq 0, \\ -1 & \text{else.} \end{cases}$$

Show how the solution in (10) can be obtained starting from (9).

In Python:

1. If you want, you are free to use any custom image, as long as you make sure to crop/reshape it to a grayscale image of size 256×256 and normalize it to $[0, 1]$[1]. Then you simply work with it as a regular `numpy.ndarray`. Else use the provided image.
2. Write functions to decompose your image into B non-overlapping blocks of 8×8 and to rearrange the blocks again into the image. Your separated blocks can be saved in a `numpy.ndarray` of shape $B \times n_B^2$, where the second dimension represents the flattened patches (you can use `numpy.ndarray.flatten`).
3. Implement the 2D DCT dictionary given in eq. (7), setting $d = 8$.
4. Run the Frank-Wolfe algorithm (set $t = 0.01$) and plot the reconstructed result as an image again. Use a sufficient number of iterations K for the algorithm to converge. Plot the input and the compressed image. Bear in mind that the iterates should not tackle the DC part (i.e. the first, constant-valued basis function) in the compression, i.e. this should remain unchanged.
5. Implement the solution of the LASSO problem in (10) and plot the results for $\lambda \in [0.01, 0.1, 1.]$.

Show and describe all your results in the report!

[1] This can be done using `skimage.io.imread` and by setting the parameter `as_gray` to `True`. For this case you can additionally import the respective module from `skimage`.