The Chinese University of Hong Kong, Shenzhen
SDS · School of Data Science

Andre Milzarek · Fall Semester 2022/23

MDS 6106 — Optimization and Modeling

Exercise Sheet 4

Assignment A4.1 (Smooth Image Inpainting): (approx. *100* points)
In this exercise, we consider the total variation image inpainting problem

$$\min_{\boldsymbol{x}\in\mathbb{R}^{mn}} \ \frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2+\mu\sum_{i=1}^{m-1}\sum_{j=1}^{n-1}\|\boldsymbol{D}_{(i,j)}\boldsymbol{x}\|_2, \quad \mu>0, \tag{1}$$

that was discussed in the lectures. Here, $\boldsymbol{x}=\mathrm{vec}(\boldsymbol{X})$ corresponds to a vectorized image $\boldsymbol{X}\in\mathbb{R}^{m\times n}$, $\boldsymbol{A}\in\mathbb{R}^{s\times mn}$ is a selection matrix that selects pixels according to a given inpainting mask, and $\boldsymbol{b}\in\mathbb{R}^s$ contains the undamaged image information. The matrices $\boldsymbol{D}_{(i,j)}$ are discretized image gradients and $\mu>0$ is a given regularization parameter.

a) On Blackboard, we have provided two datasets containing different test images and test inpainting masks. The test image package also contains two test files that showcase how to load, read, and write image files in `MATLAB` and `Python`.

   Get familiar with the code and image operations. Generate a plot of *two* damaged images based on the following instructions:

   - The original image information should be plotted in grayscale (in the background).
   - The inpainting mask (i.e., the scratches, ...) should be shown (in the foreground) using some color (like red $\mathtt{r}=[1,0,0]$ or green $\mathtt{g}=[53,130,134]/255$) to improve visibility.

   Hint: Not all of the image and mask sizes fit or match directly. You can use commands like `imresize` or `cv2.resize` to adjust sizes (if necessary). It is also possible to create masks that randomly "delete" a certain percentage of the image information.

In order to handle the nonsmoothness of the ℓ_2-norm, we substitute $\varphi(\boldsymbol{x})=\mu\sum_{i=1}^{m-1}\sum_{j=1}^{n-1}\|\boldsymbol{D}_{(i,j)}\boldsymbol{x}\|_2$ in (1) with the following simple smooth option:

$$\tilde{\varphi}(\boldsymbol{x})=\frac{\mu}{2}\sum_{i=1}^{m-1}\sum_{j=1}^{n-1}\|\boldsymbol{D}_{(i,j)}\boldsymbol{x}\|_2^2=\frac{\mu}{2}\|\boldsymbol{D}\boldsymbol{x}\|_2^2,$$

where $\boldsymbol{D}=(\boldsymbol{D}_{(i,j)})_{i,j}\in\mathbb{R}^{2mn\times mn}$ stacks all of the matrices $\boldsymbol{D}_{(i,j)}$ in a large $2mn\times mn$ matrix.

b) Show that a point $\boldsymbol{x}\in\mathbb{R}^{mn}$ is a solution of the smoothed inpainting problem

$$\min_{\boldsymbol{x}} \ f(\boldsymbol{x})=\frac{1}{2}\|\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}\|^2+\tilde{\varphi}(\boldsymbol{x}) \tag{2}$$

   if and only if $(\boldsymbol{A}^\top\boldsymbol{A}+\mu\boldsymbol{D}^\top\boldsymbol{D})\boldsymbol{x}=\boldsymbol{A}^\top\boldsymbol{b}$.

c) Implement the accelerated gradient method (Lecture L-09, slide 19) for problem (2). The extrapolation parameter and step size should be chosen as follows:

$$\alpha_k=\frac{1}{L}, \quad \beta_k=\frac{t_{k-1}-1}{t_k}, \quad t_k=\frac{1}{2}(1+\sqrt{1+4t_{k-1}^2}), \quad t_{-1}=t_0=1.$$

   Here, L denotes the Lipschitz constant of the gradient mapping ∇f. It can be shown that the constant is given by $L=1+8\mu$.

d) Implement a function `cg_method` in `MATLAB` or `Python` to solve a general linear system of equations

$$\boldsymbol{B}\boldsymbol{x} = \boldsymbol{c},$$

where $\boldsymbol{B} \in \mathbb{R}^{n\times n}$ is a symmetric, positive definite matrix and $\boldsymbol{c} \in \mathbb{R}^n$ is given. (Later, we will set $\boldsymbol{B} = \boldsymbol{A}^\top\boldsymbol{A} + \mu\boldsymbol{D}^\top\boldsymbol{D}$ and $\boldsymbol{c} = \boldsymbol{A}^\top\boldsymbol{b}$ to compute a solution of the problem (2)).

Your function should have the following input parameters: `B`, `c`, `tol`, and $\boldsymbol{x}^0$. A pseudo code of the conjugate gradient method is shown below.

Algorithm 1: The Conjugate Gradient Method

Initialization: Select an initial point $\boldsymbol{x}^0 \in \mathbb{R}^n$ and set $\boldsymbol{r}^0 = \boldsymbol{B}\boldsymbol{x}^0 - \boldsymbol{c}$, $\boldsymbol{p}^0 = -\boldsymbol{r}^0$.
for $k = 0, 1, \ldots$ **do**
  Compute the updates $\alpha_k = \frac{\|\boldsymbol{r}^k\|^2}{(\boldsymbol{p}^k)^\top\boldsymbol{B}\boldsymbol{p}^k}$, $\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha_k\boldsymbol{p}^k$, $\boldsymbol{r}^{k+1} = \boldsymbol{r}^k + \alpha_k\boldsymbol{B}\boldsymbol{p}^k$.
  If $\|\boldsymbol{r}^{k+1}\| \leq$ `tol`, then STOP and $\boldsymbol{x}^{k+1}$ is the output.
  Calculate $\beta_{k+1} = \frac{\|\boldsymbol{r}^{k+1}\|^2}{\|\boldsymbol{r}^k\|^2}$ and $\boldsymbol{p}^{k+1} = -\boldsymbol{r}^{k+1} + \beta_{k+1}\boldsymbol{p}^k$.

e) We now want to solve the smoothed inpainting problem (2) utilizing different methods:

- The accelerated gradient method implemented in part c).
- The conjugate gradient method for the linear system $(\boldsymbol{A}^\top\boldsymbol{A} + \mu\boldsymbol{D}^\top\boldsymbol{D})\boldsymbol{x} = \boldsymbol{A}^\top\boldsymbol{b}$.
- A standard solver for the linear system $(\boldsymbol{A}^\top\boldsymbol{A} + \mu\boldsymbol{D}^\top\boldsymbol{D})\boldsymbol{x} = \boldsymbol{A}^\top\boldsymbol{b}$. (You can use `B\c` in `MATLAB` or `scipy.sparse.linalg.spsolve(B,c)` in `Python`).

Test and compare your implementations using different images, masks, and sizes (use at least three different test images).

You can select $\boldsymbol{x}^0 = \boldsymbol{0}$ as initial point and `tol` $= 10^{-2}$ (or `tol` $= 10^{-4}$) for the accelerated and conjugate gradient method. The parameter μ can be set to $\mu = 0.01$. Report the required cpu-time of the different algorithms and compare the reconstruction quality of the obtained results $\boldsymbol{x}$ using the peak-signal-to-noise ratio:

$$\texttt{PSNR} = 10\log_{10}(mn/\|\boldsymbol{x} - \boldsymbol{u}\|^2)$$

Here, $\boldsymbol{u} \in \mathbb{R}^{mn}$ is a vectorized version of the original (undamaged) image $\boldsymbol{U} \in \mathbb{R}^{m\times n}$. How many iterations do the accelerated and conjugate gradient method require in general to recover an image with high `PSNR`-value (close to the true solution)? How much time can be saved by applying the conjugate gradient method?

Remark: Try to implement all methods as efficient as possible. The linear system solvers require all matrices to be stored in a `sparse` format. The test package contains code to generate $\boldsymbol{D}$ as a sparse matrix. Mimic these principles when building $\boldsymbol{A}$. If necessary, resize the images to smaller sizes if the runtime is too large or you encounter memory problems.

Sheet 4 is due on **Dec, 4th**. Submit your solutions before **Dec, 4th**, **11:00 pm**.