Numerical Algorithms for Computer Vision and Machine Learning
F. Bernard, L. Sang, F. Hofherr, Technische Universität München
Winter Semester 20/21

Matlab Submission 1

Due on 6th of December, 18:00

Exercise 1 (Inpainting)**.** In this exercise we will reconstruct a corrupted image using inpainting. The problem setting is as follows. We are given an image that contains corrupted pixels as well as the indices of the corrupted pixels. The task is to find new pixel values for the corrupted pixels to best reconstruct the pixels, see also Figure 1.

Figure 1: On the left the corrupted image. We are given the indices of the pixels containing the fence and the goal is to assign those pixels new values to reconstruct the image. The result can be seen on the right

A digital image is represented in the computer as a $\mathbb{R}^{n\times m\times n_c}$ matrix, where n are the vertical number of pixels, m are the horizontal number of pixels and n_c is the number of color channels (3 for an RGB image).

A possible approach to inpainting is to impose the following two conditions. First, neighboring pixels should be similar, second, the un-corrupted pixels should not be changed. Let f denote the corrupted image. We can write the inpainting problem as the following constrained optimization problem

$$\begin{aligned} \min_{u\in\mathbb{R}^{n\times m\times n_c}} &\sum_{i,j,c} \left(u_{i+1,j,c}-u_{i,j,c}\right)^2 + \left(u_{i,j+1,c}-u_{i,j,c}\right)^2 \\ &\text{s.t. } u_{i,j,c} = f_{i,j,c}\ \forall\,(i,j)\in\mathcal{I} \end{aligned} \tag{1}$$

Where u is the reconstructed image that we want to find. $\mathcal{I}$ is the index set of the un-corrupted pixels and we require u to be equal to the image f for those pixels, which imposes the second condition

We have incorporated the first condition that neighboring pixels need to be similar, by penalizing the forward differences in the two directions in a squared fashion. Note that boundary conditions are necessary for this approach, since e.g. the most right pixels do not have a pixel right of them. Usually, we are virtually extending the image with pixels that have the same value as the last pixels, which means that the difference for the bottom and right pixels is zero.

In the following we will reformulate the problem to an unconstrained least squares minimization which you can solve with the methods discussed in the lecture. You downloaded a corrupted image `corrupted.png` as well as a logical matrix `mask.mat` that contains 1s for the corrupted pixels and 0s for the un-corrupted pixels

General notes on the implementation:

- The matrices will become quite large. Use sparse matrices to store them in Matlab
- Use `imread` to load images, and `imshow` to display them
- Images in MATLAB are stored as `uint8`. For the computations you need to convert the matrices to `double`. Note that you need to convert them back to `uint8` if you want to display them
- The mask is loaded by `load('mask.mat')`
- Possibly useful functions: `spdiags, speye, kron`

a) (30 Points) We first reformulate the summation using a finite difference matrix. The form we want to archive is the following

$$\sum_{i,j,c} (u_{i+1,j,c} - u_{i,j,c})^2 + (u_{i,j+1,c} - u_{i,j,c})^2 = \left\| \begin{bmatrix} D_x \\ D_y \end{bmatrix} u \right\|^2 = \|Du\|^2 \tag{2}$$

To do so, we first stack the image into a (long) column vector. This is done by stacking the columns for each color channel and then stacking the color channels onto each other (in MATLAB you can just use `(:)`).

The matrix D stacks two matrices D_x and D_y that compute all forward distances in x- and y-direction (for all color channels).

Complete the function `D = createDifferenceMatrices(n, m, nc)` that takes as input the dimensions of the image and returns the finite difference matrix

$$D = \begin{bmatrix} D_x \\ D_y \end{bmatrix}$$

Include the boundary conditions as described above and make sure to remove any unnecessary rows or columns in D.

b) (35 Points) The optimization problem is still constrained. However, the constraints are fairly simple since they only tell us which part of the image not to change. We will now introduce two (sparse) matrices X and Y to decompose the optimization variable as

$$u = X\tilde{u} + Yf \tag{3}$$

Here $\tilde{u}$ contains only the unknown pixel values, while f is again the full (vectorized) corrupted image. The second term imposes the boundary conditions, by automatically assigning the un-corrupted image values to the respective entries in u.

Complete the function `[X, Y] = getDecomposition(mask, nc)` which takes as input the (logical) mask and the number of color channels and returns the decomposition matrices X and Y.

c) (35 Points) We now can write Equation (1) as an unconstrained problem in the corrupted pixel values $\tilde{u}$. Use Equation (2) and Equation (3) to write the problem in the form

$$\min_{\tilde{u}} \|A\tilde{u} - b\|^2$$

After solving this minimization, use Equation (3) to reconstruct the complete image.

Complete the function `image_inpainted = inpainting(image, mask)` that takes as input the corrupted image as well as the mask and combines all the steps that you have done so far and solves the resulting system of equations. **Return your reconstruction as valid image matrix**, i.e. reshape it and convert it back to `uint8`.

You can use the script `main_inpainting` to test your code. You will get the 35 points if your code successfully reconstructs the image.