Version: April 25, 2022

Linköping University

Name of student _____

Personal number _____

Date _____

Teaching assistant _____

Image restoration

Computer Exercise 4

1 Introduction

The goal of this exercise is to implement and evaluate methods for 2D image denoising and inpainting based on diffusion-like methods. In order to fully understand all steps, you will make a low-level implementation, where each diffusion iteration is explicitly applied to an image. The correctness of the implementation will be demonstrated by applying it on a number of images with added noise or missing data. Note that, as in all iterative numerical schemes, exact details of filters and step-lengths are important, and incorrect settings make the difference between a great result and garbage.

Take special note of the design of the Hessian in lecture 15 p.47 (references the slides from 2022) and the gradient operators in lecture 14 p.15-16.

1.1 Preparations

Before you come to the computer exercise you should have read through this exercise guide and completed the home exercises. They are all clearly marked with a pointing finger. To be able to do this you should have read and understood the material related to variational methods, image enhancement and optimization in lectures 13, 14 and 15.

Expected preparation time: 8h

1.2 Python

This lab requires Python 3.6+, *SciPy*, and *OpenCV*. On ISY computers you set up the environment by calling

```
$ module add courses/TSBB15
```

To access the lab functions from within Python use

```
>>> import lab4
```

Please familiarize yourself with the available functions in this module by calling (from a Python prompt):

```
>>> help(lab4)
```

2 Algorithm Outline for Anisotropic Diffusion

The partial differential equation (PDE) for anisotropic diffusion is defined as

$$\frac{\partial u}{\partial t} = \text{div}(D\nabla u) \tag{1}$$

where D is the diffusion tensor and ∇u denotes the gradient of image $u : \Omega \mapsto \mathbb{R}$, where Ω is the image domain. Rather than implementing the above PDE we will here consider an alternative formulation.

Preparation Question: Expand the divergence form in (1) and make the appropriate simplification to obtain the *Trace-based* diffusion equation.

Hint: What does a slow varying D mean for $\text{div}(D\nabla u)$?

In previous computer exercises you implemented the structure tensor T.

Preparation Question: What is the definition of the structure tensor and how do you obtain the diffusion tensor D from it?

Hint: How do you compute eigenvalues and eigenvectors from a 2×2 symmetric matrix?

Consult the lecture notes and prepare an outline of the algorithm for anisotropic diffusion. The plan should address

- Which operators and computations are required?
  Which ones are available, and which ones do you need to implement?
- How do you implement the computations of derivatives (gradients, etc)?
- The implementation must not include any explicit loops over the image positions since they are very time consuming. How do you avoid explicit loops?
- Which parameters to choose?
- How to test the individual steps of the algorithm?
- What data do you have for evaluation and what is the expected result?

Preparation Question: What result do you expect if you apply the method to an image that does not contain any noise?

Discuss the plan with the exercise assistant and get an approval before you start to implement it.

2.1 Implementation

Do the implementation according to your plan. Make sure to test each computational step such that they give the expected result. Make a final test of your implementation by applying it to a synthetic image that contains a white circle on a black background with a moderate amount of noise added to it, using only a single iteration of the diffusion process. For this you can use the provided function `lab4.make_circle()`.

Question: Which values appear to work best for your parameters? How sensitive are they to variations?

2.2 Parameter tuning

As most of those familiar with PDEs know, the solvers are highly sensitive to parameter values – an incorrect parameter setting will cause unstable behavior in the iterative solver.

Note: Good parameter values for your system depend on many minor implementation aspects, and thus your optimum is very likely to be different from that of your neighbour. No one else can thus provide you with the "Correct" parameter settings, and you need to tune the parameters yourself.

Question: Which parameters are not specified by the equations?

When there is no principal way to set the parameters you may need to test a large range of values. The easiest is to just evaluate the function repeatedly with different parameter values, and to use multiplicative steps, e.g. in powers of 10 to quickly find a relevant range. Keep this in mind during implementation.

2.3 Evaluation

Now it is time to evaluate the method itself, by applying it to real images (e.g. `img = lab4.get_cameraman()`) with different types of noise. The noise should be additive, of zero mean and preferably white but can be either uniform, Gaussian, or of some other suitable distribution. It can also have different signal-to-noise-ratio (SNR). The SNR is formally defined as

$$\text{SNR} = \frac{\text{Variance of signal}}{\text{Variance of noise}} \tag{2}$$

where the variance is taken over all image points. Often, the SNR is given in dB which is 10 times the 10-logarithm of the above fraction.

Write a function that adds noise to an image such that the resulting image has a given SNR in dB. The function takes the image as an input parameter. Choose a fixed type of noise or have an additional argument control the type of noise.

Question: Apply the method onto real images with additive noise of 3 different SNRs. How many iterations are needed to get an acceptable result for different noise levels?

Question: What happens if we use many more iterations than necessary? Does the result converge to the noise free image? Explain the result.

Not all types of noise are additive. For example, it is fairly common that noise can be multiplicative instead of additive. Such noise can be added by first taking the logarithm of the image intensity, then adding noise, and then taking the exponential.[1]

Question: Apply the method onto an image with multiplicative noise. What is the main difference in the result compared to additive noise? Explain the result.

3 Inpainting via Total Variation

In this exercise you will implement a simple scheme for total variation based inpainting. Let Ω be the image domain and Γ be the region with missing information in the damaged image g, then

$$\mathcal{X}_{\Omega\backslash\Gamma}(\boldsymbol{x}) = \begin{cases} 1 & \text{for } \boldsymbol{x} \in \Omega\backslash\Gamma \\ 0 & \text{for } \boldsymbol{x} \in \Gamma \end{cases} \tag{3}$$

The energy functional for this problem is defined as

$$\varepsilon(u) = \frac{1}{2\lambda} \int_\Omega \mathcal{X}_{\Omega\backslash\Gamma} \cdot (u-g)^2 \, d\boldsymbol{x} + \int_\Omega |\nabla u| \, d\boldsymbol{x} \tag{4}$$

where $\lambda > 0$ is a parameter. By computing the variational derivative (useful to do as an exercise) you will obtain the Euler-Lagrange equations

$$\begin{cases} \mathcal{X}_{\Omega\backslash\Gamma} \cdot (u-g) - \lambda \text{div}\left(\dfrac{\nabla u}{|\nabla u|}\right) = 0 & \text{for } \boldsymbol{x} \in \Omega \\ \nabla u \cdot \boldsymbol{n} = 0 & \text{for } \boldsymbol{x} \in \partial\Omega \end{cases} \tag{5}$$

where $\boldsymbol{n}$ is the normal vector to the image domain. Expanding the divergence term (also useful as an exercise), you can formulate the following iterative scheme

$$u^{s+1} = u^s - \alpha \left(\mathcal{X}_{\Omega\backslash\Gamma} \cdot (u^s - g) - \lambda \left(\frac{u^s_{xx}(u^s_y)^2 - 2u^s_{xy}u^s_x u^s_y + u^s_{yy}(u^s_x)^2}{|\nabla u^s|^3} \right) \right) \tag{6}$$

Implement the above numerical scheme by re-using code from the previous exercise and evaluate your algorithm by interpolating an image to twice its size. Alternatively, you may set a random subset of pixels to zero [2], and define a corresponding missing data mask. Note that the parameter values need to be chosen with care: If too small nothing happens, and if too large, the iteration will diverge.

[1] Note that SNR is defined for additive noise. To find the equivalent additive noise, you can use the relation $s_{\text{additive\_noise}} = s_{\text{noisy}} - s_{\text{ideal}}$.

[2] Note that it is easier to get the method to converge if a better starting point is chosen. E.g. replacing the zeroes with interpolated values helps.

Question: What do the parameters α and λ represent and what are suitable values? What number of iterations is sufficient to obtain the desired result?

Preparation Question: What is the expected behavior of the algorithm in the two regions Γ and $\Omega \backslash \Gamma$?

Extra

Question: The operator $\mathcal{X}_{\Omega\backslash\Gamma}$ defines the observed pixels. The operator can be interpreted as a projective operator $\mathcal{P}$ such that $\mathcal{X}_{\Omega\backslash\Gamma} = \mathcal{P}^*\mathcal{P}$ where $\mathcal{P}^*$ is the adjoint. Can you give an example of such an operator?

Hint: Think of an operator that describes up- and down-sampling of an image and how it relates to the definition of an adjoint operator.

In the case that you have finished and been approved on the previous exercises and there is sufficient time left, you can create your own missing regions Γ from an image of your choice. Additionally, think about how you could extend the method to process color images.