Wavelets assignment:
denoising and inpainting with wavelets

2023-2024

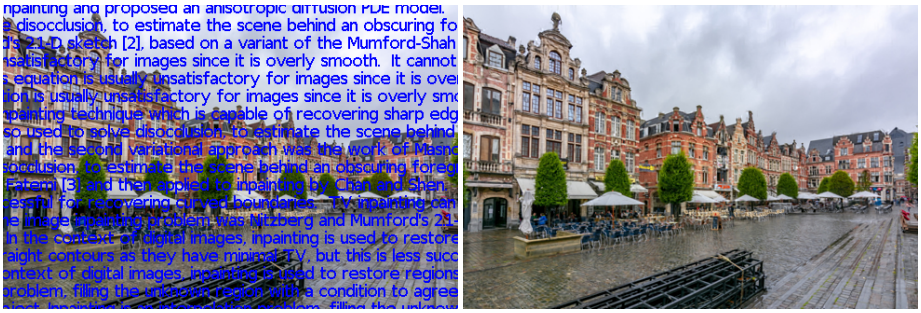

Figure 1: An inpainting algorithm at work: text that was added to a picture can be removed again with surprisingly few artifacts compared to the original figure. Missing parts of an image represent discontinuities. If the missing part is small enough, inpainting can be achieved by restoring local smoothness. Some parts of the image on the right have become fuzzier, but overall the quality is acceptable for most purposes.

1 Introduction

In the lectures we have emphasized different perspectives on wavelets. The central topic is, of course, the multiscale nature of wavelets and their time-frequency properties. Their localisation in time is adapted to their localisation in frequency – with both depending on the scale. We have examined this from the point of view of signal processing (filters and filter banks) and of approximation theory (multi-resolution analysis).

One perspective is not more ‘right’ or ‘wrong’ than the other. They are just that, perspectives. Different perspectives may lead to different insights and ultimately, apart from a better understanding on its own, to being a better practitioner.

In this project we put the theory into practice. We will contemplate the related problems of noise removal and image inpainting. All data arising from measurements come with noise, whether the measurements are produced by a cheap consumer cam-

era, a professional microphone, a hospital MRI scanner or the LIGO gravitational wave detectors. What do our perspectives have to say?

Missings parts of an image is a particularly bad case of noise. We will consider a fairly simple iterative algorithm to restore the image, in which noise removal is a basic building block. We will learn mostly by experiment, not by theory.

As we shall see, when using wavelets noise removal is related to inpainting, inpainting is related to compression, and compression is related to noise removal. If this project has a one-line summary, it could be this one: we will be concerned with discarding small wavelet coefficients – of which we hope there are many.

1.1 Preparation

Some comments before we get started:

- It will be helpful to read chapter §10 of the course notes before starting this project to develop a feel for wavelet-based applications. Particularly relevant is §10.2 (though note that we will not be using wavelet packets in this project).
- Always mention in your report which wavelet was used, and why, for any experiment. Be complete and reproducible when describing your experiments.
- Report any problems you run into with the assignment to the teacher. If something is not clear, feel free to ask. Relevant updates will appear on Toledo, visible to everyone at once.
- You are free to use any programming language, though Matlab is assumed by default.

2 Wavelet-based denoising

2.1 A univariate functions with noise

At first sight noise seems more related to the signal processing point of view than it is to function approximation. But that need not be the case, and we will first explicitly mix the two by considering a smooth function with noise. If nothing else, it may sharpen our notion of what exactly is *noise*.

To that end, consider the following piecewise smooth (and contrived) function:

$$f(x) = (2 + \cos(x))\,|x|\,\mathrm{sign}(x - 1).$$

Here, $|x|$ is the absolute value of x and sign is the sign function:

$$\mathrm{sign}(x) = \begin{cases} +1, & \text{if } x \geq 0, \\ -1, & \text{otherwise.} \end{cases}$$

We sample the function in a set of N points in the interval $[-2, 2]$,

$$f_j = f(x_j), \qquad x_j = -2 + 4\frac{j-1}{N-1}, \quad j = 1, \ldots, N.$$

Use `linspace` in Matlab instead of the formula above.

Question 2.1: Compute the wavelet transform of the vector $\mathbf{f} = \{f_j\}_{j=1}^N$, using Matlab's `wavedec` (or other software of your choice). What can you say about the

size of the coefficients? Remember to use a logarithmic scale to inspect them. Also, don't forget to mention which wavelet you have used and which boundary conditions (see `help dwtmode` and note that it is okay to use the default mode, but it is good to be aware of what the default is and even that there is a choice to be made). Feel free to substitute $f(x)$ above for another function.

We contaminate the function with noise. Useful Matlab commands are `rand` and `randn`, for uniform and Gaussian distributions respectively. Thus,

$$\tilde{f}_j = f_j + \epsilon_j, \qquad j = 1, \ldots, N,$$

where ϵ_j are noise values. For example, you could use `epsilon * rand()` to produce noise scaled by a parameter `epsilon`. The noise should be large enough to make the subsequent observations interesting, but small enough so that the denoising process afterwards still works.

Our next step is to compute the wavelet transform of $\tilde{\mathbf{f}}$, remove noise in the wavelet domain, and transform back. We may formally write this as

$$\hat{\mathbf{f}} = \Psi^{-1}(T_\delta(\Psi\tilde{\mathbf{f}})). \tag{1}$$

Here, Ψ represents the operation of taking the discrete wavelet transform of a vector, resulting in another vector which concatenates the scaling coefficients and the wavelet coefficients on all scales involved. Next, the operator T_δ represents a manipulation of the wavelet coefficients involving a parameter δ, and finally Ψ^{-1} is the inverse wavelet transform.

We let the operator T_δ act element-wise on a vector, i.e., for a vector $\mathbf{x}$ we let $T_\delta(\mathbf{x}) = \{t_\delta(x_k)\}_{k=1}^N$. There are two typical choices for the scalar function t_δ: *hard thresholding* and *soft thresholding*. See the course notes! For hard thresholding, the operation to be performed on each coefficient is:

$$t_\delta(x) = \begin{cases} 0, & |x| < \delta, \\ x, & |x| \geq \delta. \end{cases}$$

The change to coefficients close to δ in absolute value can be rather abrupt. For that reason, soft thresholding alters *all* coefficients by δ, not just the small ones:

$$t_\delta(x) = \begin{cases} 0, & |x| < \delta, \\ \operatorname{sign}(x)(|x| - \delta), & |x| \geq \delta. \end{cases}$$

Soft thresholding is not usually applied to the scaling coefficients, only to the wavelet coefficients. (Ask yourself why that might be the case.)

Task 2.2: Implement a denoising scheme based on soft and hard thresholding in the wavelet domain. Illustrate with an example. Mention the values you have used and describe the results.

You have been forced to choose a noise level `epsilon`, and a threshold parameter `delta`. For a given `epsilon`, simply experiment with `delta` until you see an interesting result and report the values with your experiments. (In fact you've made many other choices as well, and your results may very well depend on this.)

Note that in our contrived example you have a *ground truth*: you know exactly what the original function is. We could check the accuracy of the result against the ground truth. In fact, let's do that.

Question 2.3: Aim for a combination of parameters such that the denoised data is visibly smoother than the noisy data. Does this mean you also recover the accuracy of the original samples? Everywhere? Focus on the behaviour near $x = 0$, near $x = 1$ and near the edges. (Why those points?)

2.2 Images with noise

Denoising an image is more involved. Interesting questions are: which wavelet to use? Which wavelet transform? Which threshold? There aren't always precise answers, or correct-versus-wrong answers, but there is definitely a lot of variation in the results.

Unfortunately, quantifying results in image processing is not always very mathematical, but often visual – and hence subjective, though for good reason.[1] A frequently used quantitative metric to capture the difference between $\mathbf{f}$ and $\tilde{\mathbf{f}}$ is the *Signal to Noise Ratio* (SNR):

$$SNR = 10 \log_{10} \frac{\|\mathbf{f}\|_2^2}{\|\mathbf{f} - \tilde{\mathbf{f}}\|_2^2} = 10 \log_{10} \frac{\sum_k f_k^2}{\sum_k (f_k - \tilde{f}_k)^2}.$$

This quantity is reported in the decibel (dB) unit.

Use the command

```
[A,cmap]=imread(filename)
```

to read an image into Matlab.[2] Images can be shown again using

```
image(A); colormap(cmap);
```

or using the `imshow(A, cmap)` command.

Task 2.4 Implement a wavelet-based denoising scheme using the wavelet transform commands `wavedec2` and `waverec2`, for a user-given threshold. (You may want to look into what the output of the `wavedec2` and `waverec2` commands actually is.)

Question 2.5 Add noise to the image and compare soft thresholding and hard thresholding again with your algorithm. Does one give better results than the other? How did you decide that? Try several wavelets and discuss/explain your results.

For this project, it is sufficient to simply experiment with different thresholds and see what works best. But know that there is a lot of statistics and mathematics behind the choice of optimal thresholds, depending on the knowledge one has about the distribution of the noise. We are at a serious disadvantage now because there is no longer a ground truth: the image you started from may have had some noise already. What is noise in this context, anyway.

[1] A mathematician who wants to gracefully admit to using a visual criterion would say the results are measured in the *eyeball norm*, meaning that any change invisble to the naked eye might as well be zero.

[2] The dimensions of A depend on the filetype. For grayscale images, A may be an $M \times N$ matrix of unsigned integers (`uint8`), which are best converted to doubles (`A=double(A)`) before proceeding. For images with colour, A may be of size $M \times N \times 3$, where the third dimension refers to red, green and blue colour channels respectively. The simplest way to proceed is to denoise each of the channels separately in that case. Also, read the questions of this year's PC-session for more information, if you haven't done so already.

Most importantly, however, the basic question is: why do thresholding schemes even work at all? The crucial observation is that *the wavelet transform does not decorrelate noise*. In other words, the wavelet transform of a noisy signal remains a noisy signal. Thus, one can assume that all wavelet coefficients contribute equally to the noise, regardless of whether they are large or small. If there are more small coefficients than large ones, removing the small ones disproportionately removes noise rather than signal, hence improving the signal-to-noise ratio.

2.3 Using a redundant wavelet transform

Is denoising the same as compression, then, just discarding small coefficients? Not quite. The classical wavelet transform is *critically sampled*: after each filter application, the result is downsampled in order to remove mathematical redundancy. However, in the presence of noise all coefficients carry some information about the original signal. If compression is not the goal, redundancy might not be an issue. The *redundant wavelet transform* is a variant in which the downsampling step is omitted. As a result, the dimension of the output is larger than the dimension of the input: each step of the wavelet transform doubles the amount of data, as two filters are applied without downsampling. There are many ways to compute the inverse transform, since the output is now (mathematically) redundant. By averaging over several possible inverses, after you have tampered with the coefficients by thresholding, noise might be reduced further. See §7.9 of the coursenotes.

The redundant wavelet transform (also called *stationary wavelet transform*) is implemented in Matlab by the `swt2` and `iswt2` routines. You may restrict yourself to small images if the computational time becomes a burden.

Task 2.6 Implement a wavelet-based denoising scheme using the redundant wavelet transform.

Question 2.7 Compare the results of image denoising between using the standard wavelet transform and the redundant one. Discuss your findings.

Question 2.8 Go back and run your best experiment for the smooth test function in §2.1 again, this time with `swt`. Does that make a difference?

Task 2.9 Imagine the following type of noise. Say A is a matrix representing your image. Perform the commands

```
A(1:10:end,:)=0; A(:,1:10:end)=0;
```

This creates a grid of lines every tenth row and every tenth column. Attempt to remove the grid thus created by the denoising scheme. Does this work? Discuss the outcome of your experiments.

Hint: you are supposed to conclude here that it doesn't work. Inpainting will require a bit more effort.

3 Wavelet-based inpainting

3.1 Inpainting as an optimization problem

We say that a damaged image is one in which some pixel values are lost. Perhaps because they were overwritten (with grafiti as in the example on the cover), or perhaps

there were scratches, stains or cracks. *Inpainting* refers to the process of recovering the original image, by filling in the lost values. We will assume that the missing pixels are labeled, so that we can clearly distinguish between known good pixels and known missing ones.

This sounds like a daunting task. How can one possibly fill in a black spot, without having more knowledge of what was there? Indeed if the gap is large, this is clearly not possible and the iterative scheme of this project will not help. Other schemes exist based on copying other parts of the image to the missing region, in such a way that it 'fits' well. This is crude and heuristic, but sometimes effective for restoring texture.

The inpainting of smaller regions is often done by optimization of a mathematical model. A popular one is based on the concept of *Total Variation* [2]. This is useful for inpainting, as well as for denoising. Given a signal $\mathbf{x}$, total-variation based algorithms aim to find a reconstruction $\mathbf{y}$ by minimizing a functional of the form

$$TV(\mathbf{x},\mathbf{y}) = \left(\sum_k (x_k - y_k)^2\right)^{1/2} + \eta \sum_k |y_{k+1} - y_k| = \|\mathbf{x}-\mathbf{y}\|_2 + \eta\|\nabla\mathbf{y}\|_1. \quad (2)$$

The first part of this functional is an ℓ^2-norm. It ensures that the reconstruction $\mathbf{y}$ is close to $\mathbf{x}$. The second part of the functional is the ℓ^1-norm of the gradient of $\mathbf{y}$ (this is the expression $\nabla\mathbf{y}$). It is known that minimizing ℓ^1-norms promotes sparsity – see the upcoming lecture on compressed sensing for more background and evidence of this statement. If the gradient of $\mathbf{y}$ is sparse, then $\mathbf{y}$ must be piecewise constant, or more generally piecewise smooth. Note that a piecewise smooth signal has large gradients only near the jump discontinuities. Similarly, most images are piecewise smooth, with large gradients across the edges, and thus the gradient of an image is (nearly) sparse. If $\mathbf{x}$ is an image with noise or with missing gaps, we attempt to approximate it by an image $\mathbf{y}$ that fits $\mathbf{x}$ reasonably well and that is piecewise smooth. (Noise is of course not piecewise smooth, and neither are gaps – the smoothing process removes both.)

Minimizing expressions like (2) rapidly leads to having to solve large-scale nonlinear partial differential equations. Yikes! It is feasible, but complicated and computationally demanding. On the other hand, it is also known that the wavelet transform of an image is typically sparse, or nearly sparse (many coefficients are tiny). What if one minimizes the ℓ^1-norm of the set of wavelet coefficients instead? This has a similar effect of promoting sparsity. An image with a small number of significant wavelet coefficients is also piecewise smooth, like an image with a sparse gradient.

Thus, a wavelet-based alternative to (2) might be

$$R(\mathbf{x},\mathbf{y}) = \|\mathbf{x} - \Psi^{-1}\mathbf{y}\|_2 + \delta\|\mathbf{y}\|_1. \quad (3)$$

Here, $\mathbf{x}$ is a signal (or an image), $\mathbf{y}$ is a set of wavelet coefficients, $\Psi^{-1}\mathbf{y}$ represents the inverse wavelet transform of $\mathbf{y}$ and δ is a thresholding parameter. This functional can be minimized in a much easier way, as we will see below.

3.2 Formulation of an inpainting problem in Matlab

Assume an image is given by a matrix A. Say that we only know the entries of A that belong to some set Λ. This is most easily represented in Matlab using a mask matrix and *logical indexing*. Along with the matrix A, we store a matrix $mask$ of the same

size, where each entrie is a logical *true* or *false*. Such a mask can be created as follows:

```
mask = zeros(size(A));
mask(50:60,70:80) = 1;
mask = mask > 0;
A(mask) = 0;
```

The code first creates a matrix of the same size as A, containing only zeros. It sets a square of 11 by 11 pixels to one. The third statement creates a logical matrix, a matrix of boolean values. A boolean matrix can be used for indexing into a matrix, as the final line shows. At the end of these four lines, matrix A has a missing square, where 'missing' means that the values are set to 0. A mask matrix is easily visualized using the command `spy(mask)`. (Pun intended.)

The grid of Task 2.8 can be represented with logical masks as well:

```
mask(1:10:end,:)=1; mask(:,1:10:end)=1; mask = mask>0;
```

If you've already set a number of pixels to zero, then creating the corresponding mask is even easier: `mask = A == 0`.

A mask could consist of random pixels, or random squares, disks, ... A popular test is to overwrite an image with text, and then to attempt to remove that text, but it is somewhat more difficult to generate this type of mask in Matlab. The cover image of this project was generated with an online tool.

Given a matrix A and a mask that indicates the missing values, we want to restore those missing values.

3.3 An iterative algorithm

We want to minimize an expression like (3), slightly adapted to the inpainting problem at hand. In particular, when computing the match between $\mathbf{x}$ and $\mathbf{y}$ (the term with the ℓ^2-norm) we will only consider the known good pixels.

To that end, we make use of a simplified version of the algorithm presented in [1]. Though a highly specific redundant wavelet transform was constructed in that paper, we will just continue with the standard wavelet transforms we have been using for the denoising process earlier in the project.[3]

Task 3.1 Implement the following algorithm.

1. Set an initial guess B_0.
2. Iterate on n until convergence (or until you loose patience)
$$B_{n+1} = P_\Lambda A + (I - P_\Lambda)\Psi^{-1} T_\delta(\Psi B_n).$$
3. B is the output of step 2.

One piece of the formula in step 2 you'll recognize from §2.1: it is (1), the noise-removal method. A new operator here is P_Λ: it selects the entries of its argument that correspond to an index set Λ. This is what you can use the mask matrix for. The operator $(I - P_\Lambda)$ is the same, but using the complement of Λ.

What does the algorithm do?

[3]Reference [1] is included here for completeness, because it provided the original inspiration for this project, not as a source of additional information (which it isn't).

- We apply a denoising algorithm to image B_n
- Our next approximation B_{n+1} takes the known pixels from A, and fills in the missing pixels using data from the denoised version of B_n.

It is highly non-trivial to mathematically prove what this algorithm converges to, but it is fairly simple to understand intuitively. If the algorithm converges, the result matches the known pixels while smoothing out any missing regions.

Try the algorithm on a number of images with different kinds of missing information – for example randomly located squares of $T \times T$ pixels, for modest values of T. Try to come up with (experimental) answers to the following questions:

Question 3.2 Do you see a difference between soft and hard thresholding?

Question 3.3 Between a redundant and a non-redundant wavelet transform?

Question 3.4 Between different types of wavelets?

Experiment and summarize your findings. Quantify using SNR values whenever that makes sense.

4 Practical arrangements

We expect a report with the **answers to the questions above** and a brief description of the tests and experiments you have done. With each implementation, describe how you checked that the implementation is correct.

Ideally, the text of your report convinces me that you have a good understanding of the issues that are raised in all of the tasks. Include a listing of your code in an appendix, but make sure that the text itself contains sufficient information to be reproducible.

The deadline for this report is Wednesday 20 December 2023. The report may be submitted and all questions may be directed to: `daan.huybrechs@kuleuven.be`. You may work alone or in pairs of two, but the latter is recommended. Contact the teacher with any questions and keep an eye on Toledo for possible updates. Contact the teacher if you want to find a partner to team up with.

Good luck!

–Daan Huybrechs

References

[1] J.-F. Cai, R. H. Chan, and Z. Shen. A framelet-based image inpainting algorithm. *Appl. Comput. Harmon. Anal.*, 24:131–149, 2008.

[2] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D*, 60:259–268, 1992.