Wavelets assignment:
denoising and inpainting with wavelets

November 2015

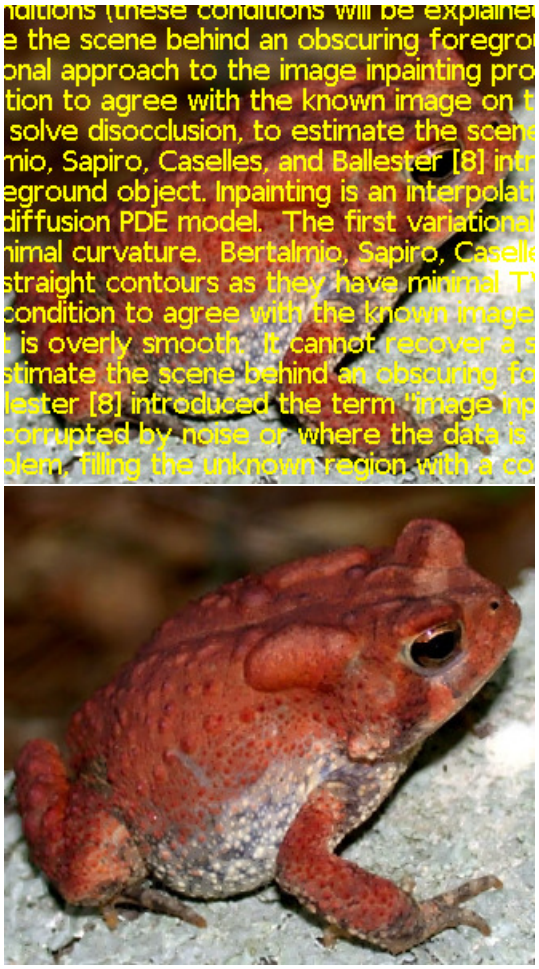

Figure 1: Modern inpainting algorithm at work: find the frog.

Introduction

Virtually all sound signals and images come with noise. Noise originates in imperfect imaging hardware, recording equipment, decaying storage, background inference... the list goes on. It is an inevitable fact of life. Some signals contain more noise than others – an old recording sounds worse than a recent CD. Still, removing noise, *denoising*, is an important operation. In fact, recent cheap cameras actually come with *worse* lenses than before, but better post-processing algorithms. Having a bad lense is okay, if you know how it warps the scene you're framing, so that you can undo the transformation after the fact. Lots of advanced signal processing techniques are applied between the time you push the button and a jpeg is stored on your camera.

It may not seem this way, but the process of *inpainting* is closely related to denoising, and that is the topic of this project. Inpainting refers to the process of reconstructing an image where parts have gone missing due to decay of the canvas. What was the original painting?

The first part of this project deals with the process of wavelet-based denoising. Usually, all information in a signal is correlated with nearby information. The colour of a pixel is closely related to the colour of its neighbors. The wavelet transform is very good at *decorrelating* signals. However, the wavelet transform will not decorrelate data that was not correlated in the first place. Thus, after a wavelet transform, noise remains noise. To be specific, imagine an image with noise. The image is correlated, the noise is not. Take the wavelet transform of the image. The wavelet transform of the image itself will generate many small coefficients, but the noise persists in all coefficients after the transform. Thus, most coefficients with a size around the noise level, are likely to be due to noise. If one removes those (by putting them to zero), one removes much of the noise without actually changing the image too much. That is the basis of wavelet-based denoising. Note how similar it is to wavelet- based compression: small wavelet coefficients are discarded. However, we intend to illustrate that, unlike for image compression, using a redundant wavelet transform is a good idea.

The second part of the project relates to inpainting of an image that has been damaged. Modern approaches for inpainting often involve solving complicated partial differential equations, with the undamaged part of the image as starting values. Surprising recent developments in research have shown that some of the complicated schemes are actually nearly equivalent to much simpler wavelet-based methods. Again, using a redundant transforms plays a key role in improving the results.

1 Wavelet-based denoising

Some comments before we get started:

- It is encouraged to read section §10.2 Image Processing of the course notes before starting this project. We will not be using wavelet packets, though.
- Run the command

  ```
  dwtmode('per')
  ```

  prior to any usage of wavelet routines in Matlab, in order to always use the periodic extension method to treat boundaries. (Or, choose otherwise and say so in your report. At the very least you should be aware of the `dwtmode` command and its implications).

- 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, ask the teacher. Updates will appear on Toledo for everyone to see.
- Feel free to use other software than Matlab's wavelet toolbox, though examples below are illustrated in Matlab code. It may be helpful for some software to restrict images such that their dimensions are powers of 2.

1.1 Warming up with an academic example

Let us start with an example in 1D which is completely artificial, so that we have control over everything. Assume a nice function is given,

$$f(x) = \exp(x),$$

so nice that it is *entire*: it is analytic everywhere in the complex plane. Entire functions are by far the easiest functions to approximate. We sample the function in a set of N points in the interval $[0, 1]$,

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

Question 1.1: Compute the wavelet transform of f_j, using Matlab's `wavedec` or other software. What can you see about the size of the coefficients? Don't forget to mention which wavelet you have used.

Let us contaminate our nice signal 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 that are large enough to make things interesting, but small enough so that the denoising process further on still works. An often used quantity to describe with one number the difference between f and $\tilde{f}$ is the *Signal to Noise Ratio* (SNR):

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

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

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

$$\hat{f} = \Psi^*(T_\delta(\Psi \tilde{f})).$$

Here, Ψ is supposed to represent the operation of taking the wavelet transform, the operator T_δ represents some manipulation of the wavelet coefficients involving a parameter δ, and finally Ψ^* is the inverse wavelet transform.

There are two typical choices for T_δ: *hard thresholding* and *soft thresholding*. 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:

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

Soft thresholding is not usually applied to the scaling coefficients, only to the wavelet coefficients.

Task 1.2: 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.

Question 1.3: The data values so far correspond to the samples of a continuous function. Given the results after denoising, how can you find an approximate value of the original function in between two samples? What would you do to evaluate $\hat{f}(x)$ at the point $x = 0.44358$? Comment on the accuracy of the result.

1.2 Moving on to images

Denoising an image is somewhat more involved. Interesting questions are: which wavelet to use? Which wavelet transform? Which threshold?

Use the command

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

to read an image into Matlab. The size of the matrix A depends 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. Feel free to denoise each of the channels separately in this case. Images can be shown again using

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

or using the `imshow` command. Read the questions of this year's PC-session for more information, if you haven't done so already.

Task 1.4 Implement a wavelet-based denoising scheme using the wavelet transform commands `wavedec2` and `waverec2`, for a user-given threshold. Let the algorithm report the output in terms of the SNR's of the noisy and denoised images, in both cases comparing to the original image without noise.

Question 1.5 Compare soft thresholding and hard thresholding with your algorithm. Does one give better results than the other? Try several wavelets and discuss/explain your results.

Task 1.6 Consider the following two strategies to find a suitable threshold:

- Write an automatic adaptive procedure that optimizes SNR – assuming that the original image is known! This is, of course, cheating a little bit. No, it is cheating a lot. But it gives an impression of best possible performance and that is what we're after.

- Find the threshold that minimize the SURE function, given by (see §10.2.2 of the coursenotes):

$$\text{SURE}(\delta) = \frac{1}{N}\sum_{k=1}^{N}(\hat{x}_{\delta k} - y_k)^2 + 2\sigma^2\frac{N_1}{N} - \sigma^2.$$

  Here, we will cheat too, since we are only learning about the topic: you may assume that the noise is normally distributed (use `randn`) and that the standard deviation σ is known and given. The values $\hat{x}_{\delta k}$ are the wavelet coefficients after the thresholding operation, y_k are the wavelet coefficients before thresholding, N is the total number of coefficients used and N_1 is the number of coefficients that was greater than the threshold δ.

Question 1.7 Is one strategy for selecting the threshold better than the other? What is the best SNR you can get?

The classical wavelet transform is *critically sampled*: after each filter application, the result is downsampled in order to remove mathematical redundancy. In the presence of noise, however, all coefficients carry information about the original signal. The *redundant wavelet transform* is one where the downsampling is omitted. As a result, the dimension of the output is possibly much larger than the dimension of the input. 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 1.8 Implement a wavelet-based denoising scheme using the redundant wavelet transform.

Question 1.9 Compare the results using the standard wavelet transform and the redundant one. Discuss your findings.

Question 1.10 Imagine the following type of noise. Say A is a matrix representing your image (or one of the RGB-colour channels). Perform the commands

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

Attempt to remove the grid thus created by the denoising scheme. Discuss the outcome of your experiments.

2 Wavelet-based inpainting

2.1 Rationale

An image that has been damaged is modelled by an image where some pixel values are lost. Often, it is in addition assumed that the image is noisy, but let's forget about that part and focus on the *inpainting* of the missing region.

This sounds like a daunting task. How could you possibly fill in a black spot if you don't have prior knowledge of what is there? For large regions, such as when you want to erase someone from a picture, the simple scheme below will not be very effective. Several schemes exist based on copying other parts of the image to the black region, in such a way that it 'fits' well. This is crude and heuristic, but sometimes effective.

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 x, total-variation based algorithms to find a reconstruction y minimize, for example, a functional of the form

$$T(x,y) = \sum_k (x_k - y_k)^2 + \lambda \sum_k |y_{k+1} - y_k| = \|x-y\|_2 + \lambda \|\nabla y\|_1.$$

The first part of this functional is an l_2-norm. It ensures that y will be close to x. The second part of the functional is the l_1-norm of the gradient of y (this is the expression ∇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 y is sparse, then 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 x is an image with noise or with missing gaps, we attempt to approximate it by an image y that fits x reasonably well and that is piecewise smooth. (Noise is of course not piecewise smooth, and neither are gaps.)

Minimizing such expressions rapidly leads to solving non-linear partial differential equations. Ouch! 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? This would promote sparsity. An analogue of the above expression would be

$$R(x,y) = \|\Psi x - y\|_2 + \delta \|y\|_1.$$

Here, x is a signal (an image), y is a set of wavelet coefficients, Ψx represents the wavelet transform of x and δ is a thresholding parameter. This functional can be minimized in a much easier way. Thus, we are looking for an image with a sparse wavelet transform that fits x reasonably well.

2.2 Matlab formulation

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 next statement creates a logical matrix. This can

be used for indexing a matrix, as the final line shows. Matrix A has now one square missing. A mask matrix is easily visualized using the command `spy(mask)`. (I am not kidding.)

The grid of last section can be represented with 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;` .

Another test is to set a number of random pixels to zero. A popular test is to overwrite an image with text, and attempt to remove that text. This is hard to achieve in Matlab.

This is the setting we will work with. The image is given by a matrix A. Some entries are zero, corresponding to the true entries of a mask, and we want to restore those values.

2.3 A simple algorithm

We want to minimize the last expression in §2.1. We will use a simplified version of the algorithm presented in [1]. Though a specific redundant wavelet transform was constructed in that paper, we will just continue with the wavelet transforms we have been using for the denoising process.

Task 2.1 Implement the following algorithm.

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

One piece of the formula in step 2 you'll recognize from §1.1. The operator P_λ restricts its argument A to the pixels that belong to the set λ. In other words:

- 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 B_n.

The point of this project is not to fully understand why algorithm 1 will converge to the minimizer of the above expression (see [1] for details), but to implement the scheme and verify that it works.

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 (experimantal) answers to the following questions:

Question 2.2 Do you see a difference between soft and hard thresholding?

Question 2.3 Between a redundant and a non-redundant wavelet transform?

Question 2.4 Between different types of wavelets?
Experiment and summarize your findings, quantify using SNR values.

3 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 16 December 2015. The report may be submitted and all questions may be directed to: `daan.huybrechs@cs.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.