

Wavelet denoising

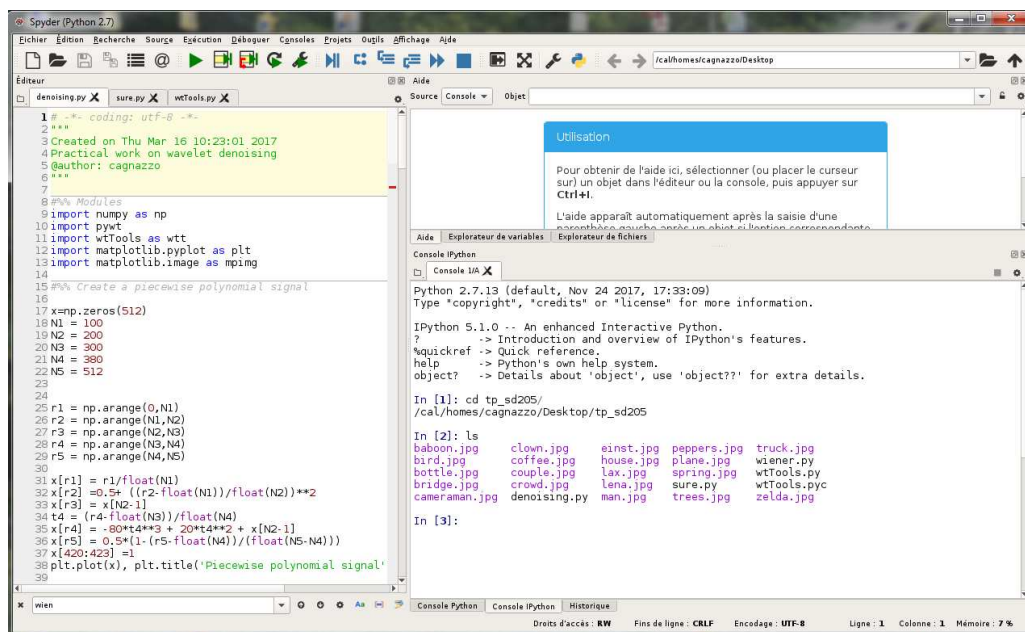
SD-TSIA205 - Advanced Statistics

09.04.2018

1 Practical work set-up

The goal of this practical work is to get familiar with wavelet transform characteristics of some simple, regular signal (1D and 2D), and then to implement and evaluate some wavelet denoising algorithm. Start by :

- Create a working directory
 - Go to the page : https://cagnazzo.wp.imt.fr/?page_id=1329
 - Download the zip file with the test images, and unzip it in your working directory
 - Download the zip file with the Python Scripts and unzip it in your working directory
 - From the terminal, launch spyder
 - In the editor, open the script denoising.py
 - Go to the working directory and type `ls`
- You should have a screen like this :



```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Thu Mar 16 10:23:01 2017
4 Practical work on wavelet denoising
5 @author: cagnazzo
6 """
7
8 %% Modules
9 import numpy as np
10 import pywt
11 import wtTools as wtt
12 import matplotlib.pyplot as plt
13 import matplotlib.image as mpimg
14
15 %% Create a piecewise polynomial signal
16
17 x=np.zeros(512)
18 N1 = 100
19 N2 = 200
20 N3 = 300
21 N4 = 380
22 N5 = 512
23
24
25 r1 = np.arange(0,N1)
26 r2 = np.arange(N1,N2)
27 r3 = np.arange(N2,N3)
28 r4 = np.arange(N3,N4)
29 r5 = np.arange(N4,N5)
30
31 x[r1] = r1/float(N1)
32 x[r2] = 0.5* ((r2-float(N1))/float(N2))**2
33 x[r3] = x[N2-1]
34 t4 = (r4-float(N3))/float(N4)
35 x[r4] = -80*t4**3 + 20*t4**2 + x[N2-1]
36 x[r5] = 0.5*(1.-(r5-float(N4))/float(N5-N4))
37 x[420:423] = 1
38 plt.plot(x), plt.title('Piecewise polynomial signal')
39
```

```
In [1]: cd tp_sd205/
/cal/homes/cagnazzo/Desktop/tp_sd205

In [2]: ls
baboon.jpg      clown.jpg      einst.jpg     peppers.jpg   truck.jpg
bird.jpg        coffee.jpg    house.jpg    plane.jpg     wiener.py
bottle.jpg      couple.jpg    lax.jpg      spring.jpg    wtTools.py
bridge.jpg      crowd.jpg     lena.jpg     sure.py       wtTools.pyc
cameraman.jpg  denoising.py  man.jpg      trees.jpg     zelda.jpg
```

The exercises are to be developed in Python. Some scripts are provided, with partially implemented methods and examples, that you have to complete. **The provided scripts are guaranteed to work ONLY ON THE SCHOOL PC, and using the “spyder” environment with Python 2.7 and pywt 0.3.0.** You may use your own PC and / or different environments or versions but in this case we do not assure we can help you in making them run.

2 Vanishing moments and DWT of 1-D signals

- We defined in class the vanishing moments (VMs) : a high-pass filter with impulse response h has M vanishing moments if, for all polynomials inputs $p[n] = \sum_{i=0}^{(M-1)} a_i n^i$, the output is null : $h * p = 0$. Another definition is that the filter absolute frequency response is null in zero with its first $M - 1$ derivatives.
- Let $h_0[0] = 1$, $h_0[1] = -1$, and $\forall n \notin \{0, 1\}, h_0[n] = 0$.
Let $\widehat{h}_0(\nu)$ be its Fourier Transform : $\widehat{h}_0(\nu) = \sum_{m \in \mathbb{Z}} h_m e^{-2i\pi\nu m}$.
 - Prove that it filters out constants
 - Prove that $|\widehat{h}_0(\nu)| = 2|\sin \pi\nu|$
 - Comment about the 2 definitions of VM
- Let $h_m = \underbrace{h_0 * h_0 * \dots * h_0}_{m \text{ convolutions}}$.
 - Prove that it filters out polynomials of degree m
 - Prove that $|\widehat{h}_m(\nu)| = 2^{m+1} |\sin \pi\nu|^{m+1}$
 - Compute numerically the derivatives of $|\widehat{h}_m(\nu)|$ for $m = 3$
 - Comment about the 2 definitions of VMs. Note that this result can be generalized to any $m \in \mathbb{Z}$
- Create a piecewise polynomial signal of degree ≤ 3 and of size $N = 512$ points
- Compute the discrete wavelet transform (DWT) (3 levels) with a filter having 4 vanishing moments (VMs). Use the Daubechies' filters, which have the smallest support for a given number of VMs : `w = pywt.Wavelet('db4')`
- Compute the frequency response of the high-pass filter `w.dec_hi` and its derivatives, and compare with the case h_3
- Show the histograms of different subbands : comment the number of null coefficients
- Repeat questions 5-7 for a filter with less VM, such as for examples `w = pywt.Wavelet('db1')`

3 DWT of Gaussian noise

- Create a white Gaussian noise (WGN) using `noiseSamples = sigma*np.random.randn(2**N)`
- Show the histogram of the noise (`plt.hist`) and compute the empirical standard deviation with `noiseSamples.std()`
- Compute the histogram of the noise DWT coefficients. Using `noiseWT = pywt.wavedec(noiseSamples, w, mode='per', level=nLevel)`, the variable `noiseWT` is the list des subbands. Transform this list into an array to compute the histogram and the empirical standard deviation.

4 1-D signal denoising

- Add WGN with STD `sigma` to the polynomial signal. Compute the SNR with `wtt.sbSNR`. `wtt.sbbSNR(x, y, lev)` computes the subband-by-subband SNR for `lev` levels of decomposition. If `lev=0` (default value), the SNR is compute on the whole signal. Open `wtTools.py` to understand this function.
- The function `wtt.coeff1Dthresh` computes the hard or soft thresholding. Open `wtTools.py` to understand its behavior.
- The universal threshold is computed as $\sigma\sqrt{2\log K_m}$, where K_m the number of detail coefficients, that is $K_m = K(1 - 2^{-J})$ for J decomposition levels.

4. How is computed the Minimax threshold in `wtt.Minimax`? This function returns the threshold normalized with respect to σ_m as a function of K_m
5. Compute the hard and soft thresholding of the noisy signal with the Minimax threshold and the universal threshold. Comment the results
6. Repetez these operations for the signal "wavy". Comment the resultat.
7. Repeat these operations with different noise STD (between 0.1 and 1) and different numbers of decomposition levels (between 2 and 5). Comment the results.

5 2-D DWT over images

1. Load one of the provided images and add WGN with $\sigma_b = 30$. Show the original and noisy image .
Use `plt.imshow` for displaying the images and `plt.set_cmap` for defining the "colormap" of the figure. For the noisy image, the values are clipped in the $[0, 255]$ interval, and casted to unsigned 8-bits integers.
2. Compute a 4-level DWT of the original and of the noisy image using the Daubechies' wavelets with 3 vanishing moments : `wav='db3'` The decomposition is performed using `wavedec2(image, filterbank, mode, level)` The mode variable defines the management of image borders. The ortogonal filters need periodization.
3. The `coeff` variable is a list containing the resolution levels of the DWT. `coeff[0]` contains the approximation subband ; `coeff[n]` is the list of the 3 detail subbands for level `n`.
Considering this, explain the function `coeffs_to_array` in the module `wtTools`
4. Show the absolute vales of the original image wavelet coefficients, using `plt.imshow` and the function `wtView` provided in the module `wtTools`.
What is the difference between these 2 visualisation modes ?¹
5. Where can the large coefficients be found ?
6. Show the histograms of the wavelet subbands. What can be observed ?
7. Estimate the noise standard deviation with the median method described in class.

6 Image denoising

1. Display the DWT of the noisy image.
2. Use the function `sbSNR` in the module `wtTools` for computing the SNR subband-by-subband and also the global SNR. Use `wtt.sbSNR(arr, arrN, NLEV, 1)` to show the subband-by-subband SNR, and use `wtt.sbSNR(arr, arrN, 0, 1)` to show the global SNR. Compare the SNR of the approximation subband and of the detail subbands with the global SNR. You can compute the SNR between `img` and `noisyImg` with `wtt.sbSNR(img, noisyImg)` or with `wtt.sbSNR(arr, arrN)`. Do you fid the same result ? Why ?
3. Denoise the image by hard and soft thresholding. Start by testing 50 values of the threshold between 0 and $T_U * 1.2$, where T_U is the universal threshold. Dispay the SNR of the denoised image as a function of the threshold. Can this be a practical method to find the best threshold value? Why?
4. Compute the universal and Minimax thresholds, and perform the hard and soft thresholding. Compare the results with the previous case and comment.
5. Repeat the denoising operations with other images and changing `sigma`, the number of decomposition levels, and the DWT filterm among ('db3', 'db4', 'db5')

1. The first function performs a global rescaling while the second rescales subband-by-subband ; moreover, the latter shows the absolute value of the coefficients, and displays in white the small coefficients and in black the large ones.

6. Open the function `sure.hybridDenoisig` in `sure.py`. Explain the behavior of this function.
7. Perform the SURE denoising on the image and compare the result with the previous cases.
8. The optimal linear filter for denoising is the Wiener filter. What is the frequency response of the Wiener filter H ?
9. Compare the threshold denoising with the optimal filter (Wiener) in terms of SNR.
10. **Undecimated DWT denoising** The undecimated consists in removing the decimation operator in the filterbank implementation of the DWT.

A simple way to implement UDWT is to create several shifted version of the input signal and then apply the ordinary DWT on them. The set of shifts should cover the range $0, 1, \dots, 2^{j_{\max}} - 1$. For each shifted image we perform a SURE (or hybrid) denoising. Then, the denoised image is the average of all the estimations obtained. The script `denoising.py` shows how to implement the shifts.

Perform the UDWT on a few test images and compare the result with the other methods.