Ural Federal University

named after the first President of Russia B.N.Yeltsin

Institute of radioelectronics and information technologies

WAVELET DECOMPOSITION OF TIME SERIES DATA

Laboratory Task no. 6

Yekaterinburg

2019

Ural Federal University
Institute of radioelectronics and information technologies

Contents

1. Introduction

The Wavelet Analysis has been established and used already for a long time, and managed to prove itself as an effective means for a solution of different tasks in adaptive time series data analysis. Wavelet transform is very similar in character to Fourier transform therefore the technique of its use reminds the spectrum methods. The main difference of these two conversions is that Fourier transform uses as basis harmonic functions of fixed frequency, and the wavelet transform – the whole class of different basic functions of different scale and localization on time (wavelets). This laboratory task provides the basics about discrete wavelet transform and its usage in decomposing time series into meaningful components.

2. Laboratory Task

The final result of laboratory task is presented as the report in form of *Jupyter*-notebook (or something similar), with all the needed commands for a complete Run of cells. Now, let's begin with time series data analysis:

1) Import into your code the following libraries:

```
import numpy as np
import numpy.random as rand
import matplotlib.pyplot as plt
import h5py
import pywt
%matplotlib inline
```

2) Create the model time series with 2 periods and noise:

```
t = np.linspace(0, 1, 1024)
f1 = 10
f2 = 50
F=np.sin(2*np.pi*f1*t)+np.sin(2*np.pi*f2*t)+0.2*rand.randn(len(t))
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, np.sin(2*np.pi*f1*t), 'b')
plt.plot(t, np.sin(2*np.pi*f2*t), 'r')
plt.show()
```

3) The work with wavelets in *Python* in most cases is done with *PyWavelets* library. The wavelet analysis is a powerful and versatile instrument, but requires a careful choice of parameters and basis function. First of all, the main parameter is the basis ("mother") wavelet. There are different families of wavelets, most of them are presented here: http://wavelets.pybytes.com. Carefully familiarize yourself with different wavelets, especially their visual forms and levels/numbers, because in most cases the final decomposed data would be quite similar in its form to the basis visual form of wavelet. But let's start with one of the most useful ones – Meyer wavelet:

```
wvlt = pywt.Wavelet('dmey')
```

4) Second, the decomposition can be achieved up to different **level**. Higher level means higher number of components and higher order of decomposition. For discrete data the highest available level is limited based on time series length, for example, in our case max level is 4:

```
pywt.dwt_max_level(len(F), wvlt) # output is 4
```

5) Third, on the edges of time interval we lack data points for wavelet analysis, thus, wavelets can extend our data through different means/modes (**mode**): simple padding with **zeroes**, **constant** extrapolation, **symmetric** and **asymmetric**, **periodic** and **periodization**, and so on. Based on the initial data and the required components different mode can provide better or worse results.
6) Fourth, decomposition for discrete wavelet analysis is represented as combination of **Approximation** (cA) and **Detail** (cD) coefficients. There is always at least one approximation and several details. Number of details is always equal chosen level of decomposition. By changing basis wavelet, level, mode and combination of grouped approximations and details different components would be achieved through wavelet decomposition.
7) Let's decompose our data with Meyer wavelet up to level 4:

```
cA4, cD4, cD3, cD2, cD1 = pywt.wavedec(F, wvlt, mode='periodization', level=4)
```

8) Note the output results – exactly one Approximation **cA4** and 4 Details from level 4 **cD4** down to level 1 **cD1**. If we select level 3 of decomposition (**level=3**), then the output should be written in form: **cA3, cD3, cD2, cD1**. It is possible to simply use one output of function, but it still should be decomposed into arrays of approximation and details. Also note, that these approximation and details are just wavelet **coefficients** and not exactly the time series data.
9) To reconstruct and group our components we use this form of code:

```
Fre = pywt.waverec((cA4, None, None, None, None), wvlt, mode='periodization')
Fre2 = pywt.waverec((None, cD4, None, None, None), wvlt, mode='periodization')
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, Fre, 'b') # this is the first period component
plt.plot(t, Fre2, 'r') # this is the second period component
plt.show()
```

10) Note the keywords **None**. In wavelet reconstruction those coefficients, that are not required for grouping, are denoted as None. It means that in first component we **only** use coefficient for approximation cA4, in second component we **only** use coefficient for the 4th detail cD4. By combining different approximation and details coefficients we would achieve different decomposition of initial time series, although, the overall form of combining coefficients is restricted.

11) On your own, make the similar decomposition and reconstruction for the level 3 of wavelet decomposition (**level = 3**). Compare the results.

12) Now repeat all of those initial steps for model data decomposition for your own **student ID variant**. Names of wavelets are presented in table below. The level of decomposition, the grouping of coefficients and every other parameter should be chosen on your own – for a better result of decomposition for two periodic components.

1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
db8	*coif3*	*haar*	*sym4*	*coif1*	*db4*	*db2*	*sym5*
9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
coif5	*haar*	*sym6*	*db6*	*sym7*	*db10*	*sym8*	*db5*
17	**18**	**19**	**20**	**21**	**22**	**23**	**24**
coif4	*sym10*	*db16*	*haar*	*db20*	*sym16*	*coif2*	*db18*

13) There are several modifications of wavelet decomposition. For example, there is a **Stationary Wavelet Transform** (**SWT**). This method provides a larger pool of approximation and detail coefficients with higher level of decomposition. To use it type in *Python*:

```
(cA5, cD5), (cA4, cD4), (cA3, cD3), (cA2, cD2), (cA1, cD1) = pywt.swt(F, wvlt, level=5)
```

14) Note the output result – we have pairs of approximation and detail coefficients. Moreover, in grouping we can use them and combine however we want, providing a vastly larger pool of combinations. Let's try to recreate our initial periodic components from model data with SWT. Also, note the normalization of data due to removing additional pairs.

```
rr1 = pywt.iswt([(cA5, cD5)], wvlt) # inverse transform for one pair
rr2 = pywt.iswt([(cD4, cD3)], wvlt) # inverse transform for one pair
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, rr1/5, 'b') # normalize by level 5 (one pair)
plt.plot(t, rr2/4, 'r') # normalize be level 4 (one pair)
plt.show()
```

15) Now repeat the SWT decomposition of this data, but specifically for your **student ID variant**. Choose all of the parameters on your own.

16) Next is the most complex type of wavelet decomposition, but with the highest number of output alterations. It is called Wavelet Packet Decomposition = **WPD**. This method is based on discrete wavelet transform, but provides a full tree of approximation and detail coefficients for the chosen level. For example, on level 4 of decomposition there are 16 combinations of approximations and details: 'aaaa', 'aaad', 'aada', 'aadd', 'adaa', 'adad', 'adda', 'addd', 'daaa', 'daad', 'dada', 'dadd', 'ddaa', 'ddad', 'ddda', 'dddd'

Compare this against DWT before, where for level 4 we have only one combination 'addd' (approximation + 3 details).

17) In WPD the decomposition is represented as binary tree (binary between ‘a’ and ‘d’), and by selecting or removing the nodes of this tree the different decomposition will prevail. The WPD procedure is started with creation of full binary tree of decomposition:

```
wp = pywt.WaveletPacket(data=F, wavelet='dmey', mode='periodization')
print([node.path for node in wp.get_level(4, 'freq')]) # output all of the binary tree nodes, ordered by their frequency width
```

18) Let’s try to delete one node and see what happens:

```
del wp['aaaa'] # deleting the most inner node
reF = wp.reconstruct() # and reconstruct the data …
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, reF, 'b') # this is something periodic, but with a lot of noise
plt.show()
```

19) If simple deletion of nodes does not yield the result, then maybe selecting few of them and grouping can provide a better result:

```
wp = pywt.WaveletPacket(data=F, wavelet='dmey', mode='periodization')
new_wp = pywt.WaveletPacket(data=None, wavelet='dmey', mode='periodization')
new_wp['aaaa'] = wp['aaaa'].data          # select the first node
new_wp.reconstruct(update=True)           # update the data inside
reF1 = new_wp.data                        # reconstruct it
new_wp = pywt.WaveletPacket(data=None, wavelet='dmey', mode='periodization')
```

```
new_wp['aaad'] = wp['aaad'].data        # select the second node
new_wp.reconstruct(update=True)         # update the data inside
reF2 = new_wp.data                      # reconstruct another one
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, reF1, 'b')          # component 1
plt.plot(t, reF2, 'r')          # component 2
plt.show()
```

20) Try to achieve similar results for your **student ID variant** with different basis wavelet and combinations of nodes. Level of decomposition can also change.

21) Now let's try to decompose some data with wavelet technique, based on wavelet basis according to your student ID variant, for different time series data.

22) First, decompose the **frequency break** time series into 2 periodic components on different time-spans:

```
t = np.linspace(0, 1, 4096)
xf = np.zeros(4096)
for i in range(0, len(t)//2):
   xf[i] = np.sin(2*np.pi*10*t[i])
for i in range(len(t)//2, len(t)):
   xf[i] = np.sin(2*np.pi*120*t[i])
plt.figure(figsize = (10, 5))
plt.plot(t, xf)
plt.show()
```

23) Next, try to estimate the exponential trend in the following noise-contaminated data:

```
t = np.linspace(0, 4, 4096)
Fexp = np.exp(-0.4*np.pi*t) + 0.2*rand.randn(len(t))
```

24) Next, simulate in *Python* the time series with 4 different harmonic periods and decompose it into those 4 periodic components with wavelets:

$$u(t) = \sin\left[2\pi t(f_1)\right] + \sin\left[2\pi t(f_2)\right] + \sin\left[2\pi t(f_3)\right] + \sin\left[2\pi t(f_4)\right] + \xi(t)$$

25) At last, load the time series data from file **doppler.mat**:

```
file = h5py.File('doppler.mat','r')
data = file.get('data')
data = np.array(data)
plt.figure(figsize = (10, 5))
plt.plot(data, 'k')
plt.show()
```

26) Decompose this data into some meaningful components with wavelet. Try to de-noise this data, i.e. clear the time series from noise completely with wavelets. Use only the basis wavelet for your student ID variant, everything else (level, grouping and so on) is adjustable.

3. Report requirements

Report in Jupyter-notebook should include: number of laboratory task, name of student, student ID or variant number, student group number and the complete task (code, plots and figures) with student comments in the report.