МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени
первого Президента России Б. Н. Ельцина»

ПРИМЕНЕНИЕ ВЕЙВЛЕТ МЕТОДОВ ДЛЯ ДЕКОМПОЗИЦИИ ВРЕМЕННЫХ РЯДОВ

Методические указания к выполнению практического задания № 6

Екатеринбург
2020

Уральский федеральный университет имени первого Президента России Б.Н.Ельцина

Содержание

Введение

Целью данной лабораторной работы является знакомство и изучение средств *Python* для работы с вейвлетами. Студенты приобретут навыки по использованию этого инструмента для анализа и декомпозиции временных рядов, а также получения оценки их частотно-временных характеристик.

1. Задание на лабораторную работу

Результатом выполнения лабораторной работы является оформленный отчет в виде *Jupyter*-тетради, в котором должны быть представлены и отражены все нижеперечисленные пункты:

1) Сначала импортируйте в свой код нужные библиотеки, функции и т.д.

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

2) Создадим зашумленный временной ряд с 2 периодиками:

```
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) Наиболее широкой библиотекой для работы с вейвлетами в *Python* является *PyWavelets*. При работе с вейвлетами надо понимать, что это очень гибкий инструмент со множеством параметров. Во-первых, самым главным параметр вейвлет-разложения является сам материнский (базисный) вейвлет. Множество доступных в библиотеке *PyWavelets* семейств вейвлетов и их модификаций доступно на сайте: http://wavelets.pybytes.com. Внимательно изучите эти семейства, так как выбор базового вейвлета существенно влияет на конечный результат декомпозиции по принципу самоподобия выделенных компонент и выбранного вейвлета.

   Для начала используем вейвлет Мейера:

   wvlt = pywt.Wavelet('dmey')

4) Во-вторых, декомпозиция с помощью дискретных вейвлетов происходит до определенного уровня (**level**), ограниченного размером доступных данных. В нашем случае возможно разложение до 4 уровня:

   pywt.dwt_max_level(len(F), wvlt) # будет выведено число 4

5) В-третьих, по краям временного интервала вейвлет может по-разному трактовать конструируемые точки для экстраполяции (**mode**): простое дополнение нулями, константами, симметрично/асимметрично, периодически и т.д. В зависимости от вида исходных данных, лучше подходит тот или иной режим.

6) Ну и наконец, декомпозиция всегда происходит в виде комбинации коэффициентов **Аппроксимации** (*cA*) плюс **Детали** (*cD*). Всегда есть одна аппроксимация, а число деталей равно уровню декомпозиции. Меняя выбор группировки коэффициентов аппроксимации и деталей, будут меняться восстановленные компоненты и соответствующая декомпозиция ряда.

7) Разобьем наш исходный ряд на компоненты с помощью вейвлета Мейера, в режиме периодизации, до 4 уровня декомпозиции:

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

8) Обратите внимание, как выглядят выходные результаты: одна аппроксимация **cA4** с номером уровня и 4 детали с убывающими номерами от 4 уровня **cD4** до 1 уровня **cD1**. Если бы мы декомпозировали ряд на 3 уровня (**level=3**), то тогда выходные значения следовало бы записать как: **cA3, cD3, cD2, cD1**. Можно на выходе функции использовать и одну переменную, но ее все равно придется разбивать на отдельные элементы. Также следует помнить, что в результате декомпозиции получаются не новые временные ряды, а только вейвлет-коэффициенты декомпозиции.
9) Восстановим две периодики исходного модельного ряда:

```
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') # это будет первая периодика
plt.plot(t, Fre2, 'r') # это будет вторая периодика
plt.show()
```

10) Снова обратите внимание на форму записи восстанавливаемых компонент. Те коэффициенты, которые мы не хотим использовать для реконструкции компонент (лишние детали и т.д.), мы заменяем на **None**. Меняя комбинации используемых и неиспользуемых аппроксимаций и деталей, мы будем получать разные восстановленные компоненты.
11) Проведите аналогичную декомпозицию для 3 уровня (**level = 3**). Вейвлет не меняйте. Сравните полученные результаты.

12) Повторите все проделанные шаги по декомпозиции ряда и восстановлению его компонент **для своего варианта** базисного вейвлета. Наименование вейвлета указано в таблице ниже. Используйте любой возможный уровень декомпозиции – выберите тот, который окажется наиболее точным среди них.

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) Есть и другие модификации простого вейвлет-преобразования, позволяющие декомпозировать временные ряды. Например, есть **Стационарное Вейвлет Преобразование** (Stationary Wavelet Transform = **SWT**). Этот метод дает гораздо большие возможности декомпозиции по уровню и по комбинации аппроксимаций и деталей:

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

14) Как видно, вместо одной аппроксимации и множества деталей мы теперь получаем пары коэффициентов аппроксимации и детали. Причем, на самом деле, при восстановлении их можно использовать уже в совершенно разных комбинациях. Восстановим искомые периодические компоненты следующей комбинацией пар, при этом нам еще потребуется нормировка, так как часть коэффициентов мы полностью выбросили:

```
rr1 = pywt.iswt([(cA5, cD5)], wvlt)
rr2 = pywt.iswt([(cD4, cD3)], wvlt)
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, rr1/5, 'b') # перенормируем
plt.plot(t, rr2/4, 'r') # перенормируем
plt.show()
```

15) Проделайте декомпозицию SWT **для своего варианта** вейвлета. Уровень декомпозиции может отличаться.

16) Ну и остался самый мощный инструмент вейвлет-декомпозиции, называемый **Пакетной Вейвлет Декомпозицией** (Wavelet Packet Decomposition = **WPD**). Этот метод опирается на дискретное преобразование, но позволяет произвести полный перебор комбинаций коэффициентов аппроксимации **a** и деталей **d**. Например, для 4 уровня это будет 16 комбинаций вида: 'aaaa', 'aaad', 'aada', 'aadd', 'adaa', 'adad', 'adda', 'addd', 'daaa', 'daad', 'dada', 'dadd', 'ddaa', 'ddad', 'ddda', 'dddd' Для сравнения, на 4 уровне обычной дискретной вейвлет декомпозиции из пункта 7 доступна только одна комбинация 'addd'.

17) При таком большом числе возможных вариантов, декомпозиция представляется в виде бинарного дерева, выборка или удаление узлов из которого и будет менять после реконструкции полученные временные ряды. Но для начала создадим общую пакетную декомпозицию:

```
wp = pywt.WaveletPacket(data=F, wavelet='dmey', mode='periodization')
print([node.path for node in wp.get_level(4, 'freq')]) # выводим все комбинации узлов, упорядоченные по их частотной ширине спектра
```

18) Попробуем удалить один из узлов и посмотреть, что получится:

```
del wp['aaaa'] # удалим самый «глубокий» узел
reF = wp.reconstruct() # и восстановим ряд …
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, reF, 'b') # получим нечто периодическое, плохого качества
plt.show()
```

19) Если удаление узлов не приводит к желаемым результатам, возможно есть смысл делать отдельную выборку ветвей этих узлов:

```
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         # выбираем первую ветвь
new_wp.reconstruct(update=True)          # обновляем данные
reF1 = new_wp.data                       # восстанавливаем под нее ряд 1
new_wp = pywt.WaveletPacket(data=None, wavelet='dmey', mode='periodization')
new_wp['aaad'] = wp['aaad'].data         # выбираем вторую ветвь
new_wp.reconstruct(update=True)          # обновляем данные
reF2 = new_wp.data                       # восстанавливаем под нее ряд 2
plt.figure(figsize = (10, 5))
plt.plot(t, F, 'k')
plt.plot(t, reF1, 'b')        # компонента 1
plt.plot(t, reF2, 'r')        # компонента 2
plt.show()
```

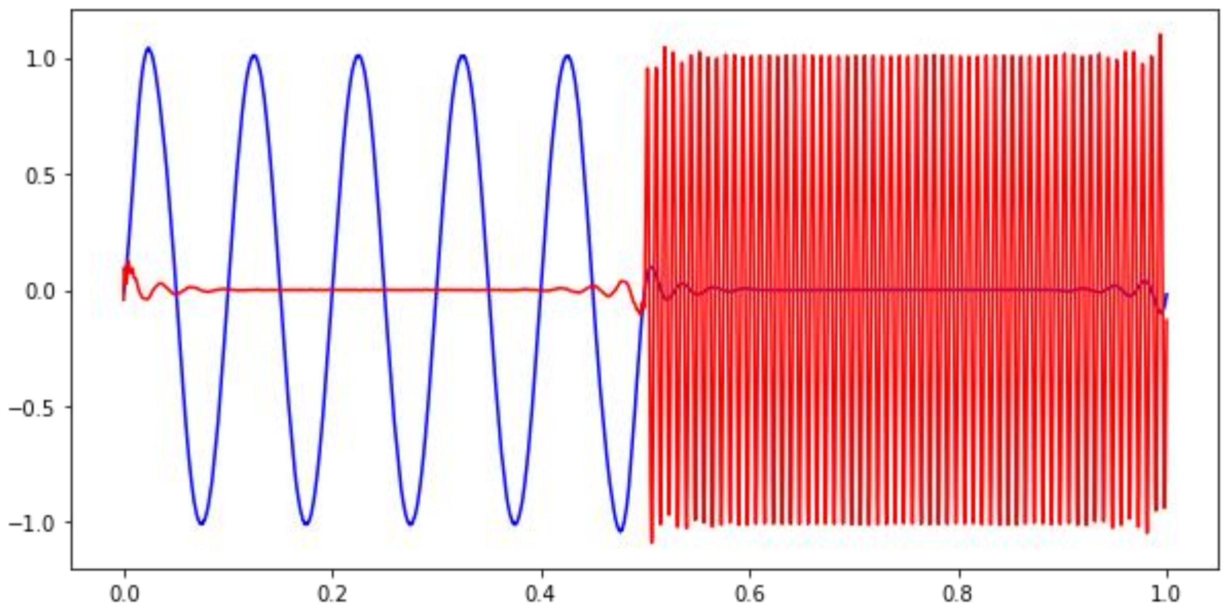
20) Проделайте подобную декомпозицию WPD **для своего варианта** вейвлета с выбором/удалением нужных узлов. Уровень декомпозиции может отличаться.

21) Теперь примените полученные навыки вейвлет-декомпозиции всех видов, на основе базисного вейвлета **для своего варианта** из таблицы, для получения компонент следующих временных рядов.

22) Декомпозируйте сигнал с частотным изломом на 2 периодические компоненты, разделенные по времени:

```
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()
```

Хороший результат выглядит примерно так:



23) Попробуйте выделить **экспоненциальный тренд** из следующего зашумленного временного ряда:

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

Внимание! Для выделения тренда опцию **mode='periodization'** внутри функций вейвлетов нужно будет убрать, так как для тренда не нужен режим периодического дополнения точек по краям.

24) Смоделируйте временной ряд из **4 гармоник с шумом**, и разделите его на **4 гармоники** с помощью вейвлет декомпозиции:

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

```
t = np.linspace(0,1,1024)
f1 = 10
f2 = 40
f3 = 100
f4 = 150
F=2.0*np.sin(2*np.pi*f1*t)+1.5*np.sin(2*np.pi*f2*t)
    +0.8*np.sin(2*np.pi*f3*t)
    +0.5*np.sin(2*np.pi*f4*t)+0.2*rand.randn(len(t))
plt.figure(figsize = (10, 15))
plt.subplot(5,1,1)
plt.plot(t, F, 'k')
plt.subplot(5,1,2)
plt.plot(t, 2.0*np.sin(2*np.pi*f1*t), 'b')
plt.subplot(5,1,3)
plt.plot(t, 1.5*np.sin(2*np.pi*f2*t), 'r')
plt.subplot(5,1,4)
plt.plot(t, 0.8*np.sin(2*np.pi*f3*t), 'g')
plt.subplot(5,1,5)
plt.plot(t, 0.5*np.sin(2*np.pi*f4*t), 'm')
plt.show()
```

Внимание! Здесь Вам потребуется выделять более двух компонент, из которых только одна будет определяться на основе коэффициента аппроксимации, а остальные – на основе коэффициентов детализации. Но коэффициенты детализации низких уровней имеют 2^n больше отсчетов (из-за квадратичной двоичной фильтрации), чем самый верхний уровень. Как тогда восстановить ряд? Тогда нужно просто использовать при восстановлении меньше **None**, чем необходимо. Например, для 5 уровня можно построить более 6 компонент (аппроксимация + 5 деталей + комбинации деталей) следующим образом:

```
cA5, cD5, cD4, cD3, cD2, cD1 = pywt.wavedec(F, wvlt, mode='periodization', level=5)
pywt.waverec((cA5, None, None, None, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD5, None, None, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD4, None, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD3, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD2, None), wvlt, mode='periodization')
pywt.waverec((None, cD1), wvlt, mode='periodization')'periodization')
```

или разные комбинации деталей:

```
pywt.waverec((None, cD4, cD3, None, None), wvlt, mode='periodization')
pywt.waverec((None, cD4, cD3, cD2, None), wvlt, mode='periodization')
pywt.waverec((None, cD3, None, cD1), wvlt, mode='periodization')
```

и т.д.

25) Оформите итоговый отчет о проделанной работе.

2. Требования к оформлению отчета

Отчет в Jupyter-тетради должен обязательно содержать: номер лабораторной работы, ФИО студента, номер варианта (либо студенческий номер), номер группы, результаты выполнения работы с комментариями студента (комментарии пишутся после #) и изображениями.