Введение

В современном мире специалистам практически в любой отрасли рано или поздно приходится сталкиваться с анализом данных предоставленных каким-либо процессом. Но на любой процесс могут воздействовать внешние, искажающие факторы. Таким образом, рассматривая данные, в виде графика или числовых значений, мы всегда должны помнить, что перед нами результат, отражающий взаимодействие множества разнообразных процессов.

Анализировать такие искажённые данные довольно сложно, а зачастую очень важно дать наиболее точный прогноз относительно какого-либо события основываясь непосредственно на предоставленных данных, отсюда и вытекает желание разделить наблюдаемый процесс на составляющие компоненты и анализировать каждую из них по отдельности. Изучая вклад каждой из таких компонент в исследуемый процесс, можно составить более четкую картину происходящего и повысить точность прогноза.

Целью данной работы будет ознакомиться с методом эмпирической декомпозиции мод и альтернативными методами разделения имеющихся данных на составные части, а так же реализовать соответствующий алгоритм на языке Python и провести тесты.

1. Методы анализа нестационарных сигналов

Традиционные методы анализа данных, как правило, предназначены для линейных и стационарных сигналов. Но многие естественные физические процессы в той или иной мере являются нелинейными и нестационарными, поэтому в последние десятилетия начали активно развиваться методы анализа нестационарных данных, такие как вейвлетный анализ, спектрограмма и преобразование Гильберта-Хуанга.

1.1 Основные понятия

Формально, термин «декомпозиция» означает разделение чего-либо на составные части, но в областях, связанных с анализом различных сигналов, последовательностей или процессов, этот термин предполагает более широкое значение. Под декомпозицией стоит понимать разложение не на реальные составляющие, входящие непосредственно в исходные данные, а на приближенные функции, которые не присутствовали в процессе формирования этих исходных данных. Такие, искусственным образом полученные, функции, не смотря на их происхождение, позволяют произвести более точный и глубокий анализ.

Нестационарные сигналы – сигналы, характер которых изменяется с течением времени.

Огибающая сигнала – функция, построенная по некоторым характерным точкам данного сигнала. В ходе данной работы, для построения огибающих, будут выбраны точки максимума и минимума сигнала.

Сплайн – отрезок, часть чего-либо.

Главное условие интерполяции (ГУИ) – построенная приближающая функция $g(x)$ должна совпадать во всех узлах с исходной функцией f.

1.2 Различные методы декомпозиции данных

На сегодняшний день существует большое количество методов декомпозиции анализируемых данных. Они отличаются подходами, лежащими в их основе, степенями сложности и областями, в которых они применяются.

1.2.1 Преобразование Фурье

Одним из известных способов декомпозиции данных является преобразование Фурье. Это преобразование принадлежит классу ортогональных преобразований с фиксированным базисом гармонических функций. Результатом преобразования Фурье является исходный сигнал, разложенный на гармонические функции фиксированной частоты и амплитуды. Стоит отметить, что преобразование Фурье всегда производится в фиксированном базисе ортогональных функций, который не зависит от характера преобразуемого сигнала. Ещё одной важной особенностью данного преобразования являются неизменные значения частоты и амплитуды для результирующих компонент. Это означает то, что если характер исходного сигнала будет изменяться с течением времени, то эти изменения не будут учтены в результатах. В данном случае, результаты будут отражать только усредненные значения. В силу своих особенностей, данное преобразование не подходит для анализа нестационарных сигналов.

Преобразование Фурье функции можно задать следующей формулой:

$$X(f) = \int_{-\infty}^{+\infty} x(t) e^{-j2\pi f t} dt, \qquad (1)$$

где $x(t)$ – некоторая функция;

j – мнимая единица;

f – переменная частота (Гц);

Величина $e^{-j2\pi ft} = \cos(2\pi ft) + j * \sin(2\pi ft)$.

1.2.2 Спектрограмма

Спектрограмма является ещё одним из основных методов и представляет собой не что иное, как ограниченный спектральный анализ Фурье по ширине окна. Последовательно сдвигая окно вдоль оси времени, можно получить распределение по частоте. Поскольку он опирается на традиционный спектральный анализ Фурье, нужно считать данные кусочно-стационарными. Но это не всегда оправдано в случае анализа нестационарных сигналов. Невозможно гарантировать, что размер окна всегда совпадает со стационарными участками сигнала. Так же существуют трудности практического применения данного метода: для локализации события во времени ширина окна должна быть узкой, но, с другой стороны, частотное разрешение требует более длинных временных рядов. Использование данного метода возможно только в определённых ситуациях, однако его легко реализовать с помощью быстрого преобразования Фурье. Большинство применений данного метода предназначены для качественного отображения анализа речи

1.2.3 Вейвлет-преобразование

Другое преобразование, позволяющее избежать проблем, которые возникаю при анализе нестационарных сигналов с помощью преобразования Фурье, - вейвлет-преобразование. Это преобразование, так же как и преобразование Фурье, производит декомпозицию в фиксированном базисе, с одним лишь отличием, что этот базис должен быть выбран перед началом преобразования. При использовании вейвлет-преобразования, каждая найденная компонента имеет параметры, определяющие её значения частоты и амплитуды, что позволяет анализировать нестационарные сигналы.

И преобразование Фурье и вейвлет-преобразование пользуются большой популярностью в силу своей универсальности и использованию в них хорошо обоснованных математических методов. Но более удобным для практического применения было бы преобразование, позволяющее работать с нестационарными сигналами и использующее базис преобразования, определяемый исходными данными.

1.3 Метод эмпирической модовой декомпозиции

Метод эмпирической модовой декомпозиции (Empirical Mode Decomposition, EMD) является составной частью преобразования Гильберта-Хуанга (Hilbert-Huang transform, HHT), предложенного Норденом Хуангом в 1995 году и предназначавшегося для исследования поверхностных волн тайфунов.

В отличие от преобразования Фурье и вейвлет-преобразования в процессе эмпирической модовой декомпозиции происходит разложение на эмпирические моды, которые не заданы аналитически и определяются исключительно самой анализируемой последовательностью. При этом базисные функции преобразования формируются адаптивно, непосредственно из входных данных. Выделенные, в процессе данного метода, эмпирические моды должны удовлетворять следующим условиям:

1. Количество экстремумов эмпирической моды и количество пересечений нуля не должны отличаться более чем на единицу.
2. В любой точке эмпирической моды среднее значение огибающих, определённых локальными максимумами и локальными минимумами, должно быть нулевым.

Удовлетворение данным условиям позволяет применить к полученным модам методы преобразования Гильберта.

1.4 Алгоритм эмпирической модовой декомпозиции

В основе алгоритма лежит построение гладких огибающих по точкам экстремума данной на вход последовательности $x(t)$ и дальнейшее вычитание этих огибающих из исходных данных. Необходимо найти все точки максимума и минимума, а затем, с помощью интерполяции кубическими сплайнами построить верхнюю и нижнюю огибающие соответственно. Далее по полученным огибающим вычисляется среднее значение $m_1(t)$, которое вычитается из исходной последовательности. На рисунке 1 представлен пример сигнала с построенными верхними и нижними огибающими.

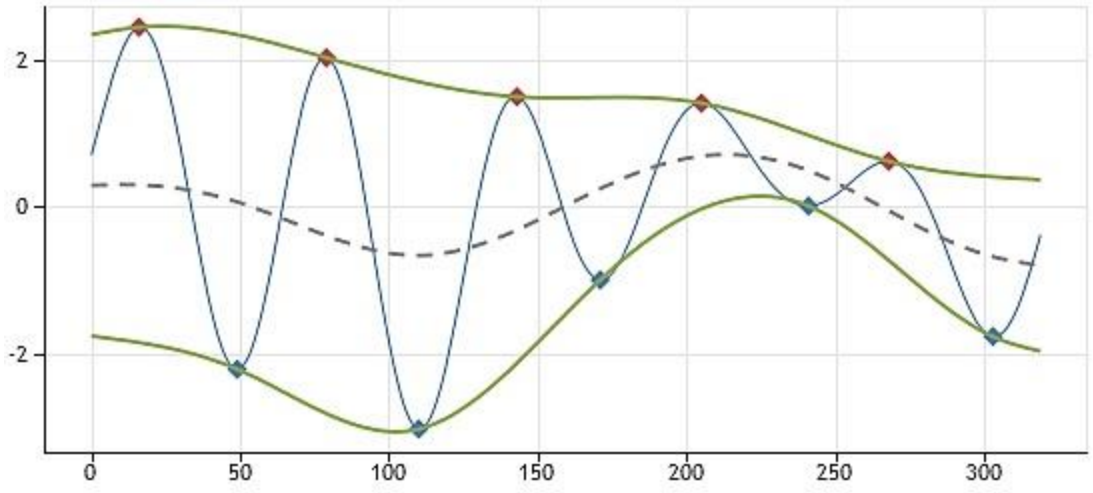


Рисунок 1. Пример построенных верхних и нижних огибающих

На рисунке красным обозначены точки максимума, синим – точки минимума, зелёными линиями - верхняя и нижняя огибающие соответственно и пунктирной линией обозначено среднее значение между огибающими.

В результате перечисленных шагов находится искомая эмпирическая функция в первом приближении $h_1(t) = x(t) - m_1(t)$. Для нахождения полноценной эмпирической моды функции, необходимо повторять действия, описанные выше, до наступления условия остановки.

1.4.1 Условие остановки алгоритма

Первым условием для остановки алгоритма, который Хуанг применил на практике, было сравнение нормированного квадрата разности между двумя последовательными итерациями алгоритма (между последней и предпоследней найденными эмпирическими модами) с наперёд заданным малым числом.

$$SD_k = \frac{\sum_{t=0}^{T} |h_{k-1}(t) - h_k(t)|^2}{\sum_{t=0}^{T} h_{k-1}^2} \quad (2)$$

где SD_k – коэффициент для сравнения;
$h_{k-1}(t)$ – предыдущее значение эмпирической моды;
$h_k(t)$ – текущее значение эмпирической моды.

Если этот параметр оказывался меньше заданного заранее числа, то алгоритм останавливался. Но у данного критерия есть несколько недостатков, хотя он и является строгим с математической точки зрения:

1. Непонятно, насколько малое значение выбирать для сравнения;
2. Данный критерий не имеет отношения к свойствам получаемых эмпирических мод, и его выполнение не гарантирует, что число экстремумов и число нулей будут равны или будут отличаться на единицу;
3. При выборе очень маленького критерия для сравнения серьезно увеличиваются вычислительные затраты, т.к. критерий необходимо пересчитывать на каждом шаге алгоритма.

Указанные выше недостатки привели к созданию более простого критерия. Он основан на простом ограничении числа итераций. Достаточно менее 10 итераций для получения компонент со свойствами, близкими к эмпирическим модам.

После наступления критерия остановки, последнее значение полученное значение $h_k(t)$, принимается за наиболее высокочастотную функцию, которая

непосредственно входит в состав исходного сигнала $x(t)$. Низкочастотные составляющие можно получить путём вычитания полученного значения из исходной функции.

$$r_1(t) = x(t) - \ h_k(t) \qquad (3)$$

Полученный остаток $r_1(t)$ принимается как новая последовательность $x(t)$, и продолжает обрабатываться по алгоритму, пока остаток $r_k(t)$ не превратится в монотонную последовательность. Таким образом достигается декомпозиция данных в $n-$эмпирическом приближении.

Ниже представлена схема алгоритма эмпирической модовой декомпозиции.

Рисунок 2. Схема алгоритма эмпирической модовой декомпозиции

1.5 Задача интерполяции

В описании алгоритма, для построения верхней и нижней огибающих использовался метод интерполяции кубическими сплайнами. Это один из методов решения задачи интерполяции. Рассмотрим подробнее.

Предположим, что у нас имеются значения функции в некоторых узловых точках. Этого недостаточно для построения графика или же анализа это функции. Необходимо найти значения в промежуточных точках. Для этого нам необходимо построить некоторую функцию (интерполянту), значения которой будет совпадать в узловых точках с имеющимися у нас значениями. Существует множество вариантов нахождения интерполянты, их отличают математические законы, на которых они основываются, и степень точности получаемых в результате значений. Интерполяция кубическими сплайнами даёт необходимую точность.

1.5.1 Интерполяция кубическими сплайнами

Для нахождения промежуточных значений необходимо построить на каждом отрезке $[x_{i-1}, x_i]$ кубический многочлен следующего вида:

$$S_i(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3, \forall\, i \text{ от } 1 \text{ до } n \quad (4)$$

Полученный многочлен называется кубическим сплайном. Для того чтобы определить сплайны $S_i(x), \forall\, i$ от 0 до n, необходимо найти значения их коэффициентов a_i, b_i, c_i, d_i (всего $4n$ штук).

Найдём эти коэффициенты из следующих условий:

1. Потребуем, чтобы сплайны, определяемые по формуле (4) удовлетворяли ГУИ.

$$\begin{cases} S_i(x_i) = f_i \\ S_i(x_{i-1}) = f_{i-1} \end{cases} \rightarrow$$

$$\rightarrow \begin{cases} a_i + b_i * h_i + c_i * h_i^2 + d_i * h_i^3 = f_i, i \text{ от } 1 \text{ до } n & (5) \\ a_i = f_{i-1}, i \text{ от } 1 \text{ до } n & (6) \end{cases}$$

где $h_i = x_i - x_{i-1}$

Отсюда получим *2n* уравнений, чего недостаточно для нахождения всех коэффициентов.

2. Получим недостающие уравнения, потребовав непрерывности первой и второй производных кубического многочлена в узловых точках.

$$\begin{cases} S_i'(x_i) = S_{i+1}'(x_i) & (7) \\ S_i''(x_i) = S_{i+1}''(x_i) & (8) \end{cases}$$

Если расписать данную систему в терминах коэффициентов сплайнов, то получится следующее:

$$\begin{cases} b_i + 2 * c_i * h_i + 3 * d_i * h_i^2 = b_{i+1}, i \text{ от } 1 \text{ до } n-1 & (9) \\ 2 * c_i + 6 * d_i * h_i = 2 * c_{i+1}, i \text{ от } 1 \text{ до } n-1 & (10) \end{cases}$$

Из этой системы получим ещё *2n-2* уравнений. Всего на данный момент имеется *4n-2* уравнения, чего все ещё не хватает для нахождения всех коэффициентов.

3. Недостающие два уравнения сконструируем из условия нулевой кривизны сплайна $S_1(x)$ в точке x_0 и сплайна $S_n(x)$ в точке x_n.

Потребуем следующее:

$$\begin{cases} S_1''(x_0) = 0 \\ S_n''(x_n) = 0 \end{cases} \rightarrow \begin{cases} 2 * c_1 + 6 * d_1 * (x - x_0) = 0 & (11) \\ 2 * c_n + 6 * d_n * (x_n - x_{n-1}) = 0 & (12) \end{cases}$$

Таким образом, уравнения 5-12 дают нам *4n* уравнений, которые представляют собой систему линейных алгебраических уравнений относительно искомых коэффициентов. Решив эту СЛАУ любым доступным способом, получим значения всех коэффициентов кубических сплайнов, тем самым однозначно определив кусочно-непрерывную интерполянту.

2. Реализация алгоритма эмпирической декомпозиции мод

Алгоритм эмпирической декомпозиции мод был реализован на языке Python 2.7, с использованием вспомогательных библиотек:

1. Tkinter – для реализации GUI
2. xalglib – для нахождения кубических сплайнов
3. matplotlib.pyplot – для построения графиков эмпирических мод

Ниже представлены основные методы программы с пояснениями.

Для удобного хранения входных данных был реализован класс Pair, хранящий индекс (x координату) и само значение (y координату).

2.1 Описание методов

2.1.1 Получение огибающей сигнала

В основе данного метода лежит вспомогательная функция spline1dconvcubic из библиотеки xalglib. Данная функция принимает на вход три параметра: массив индексов исходного сигнала, массив значений исходного сигнала и массив индексов для поиска значений интерполянты. Возвращаемое значение данной функции – массив чисел с плавающей точкой. Важное условие работы данный функции – количество точек экстремума в исходных данных должно быть не меньше двух.

Метод find_interpolant принимает на вход массив точек экстремума, разделяет его на массив индексов и массив значений, формирует массив новых индексов для поиска значений интерполянты, а после – формирует выходной массив хранящий значения типа Pair().

```
def find_interpolant(listExtremum):
    …
    newValues = xal.spline1dconvcubic(indexes, values, newIndexes)
    …
    return returnArray
```

2.1.2 Нахождение эмпирических мод

Для удобства реализация алгоритма эмпирической декомпозиции мод была разделена на несколько методов.

Метод inner_alg принимает на вход массив объектов типа Pair(), находит в данном массиве точки минимума и максимума, верхние и нижние огибающие и среднее значение между ними, а так же разницу между исходным сигналом и полученным средним значением огибающих. Данный метод возвращает массив объектов типа Pair(), хранящий разницу между исходным сигналом и средним значением огибающих.

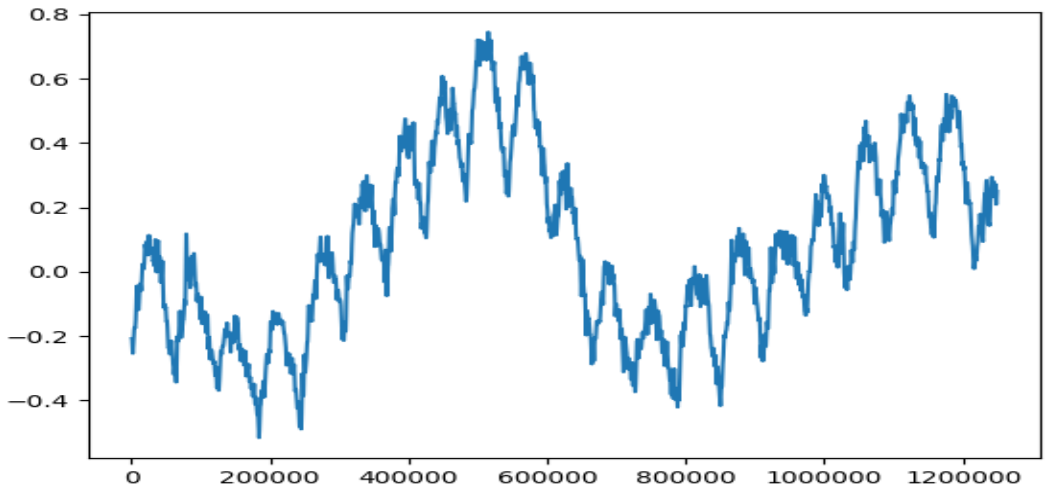
Метод process_alg принимает на вход массив объектов типа Pair() и вещественное число – значение коэффициента для условия остановки алгоритма. Внутри данного метода используется метода inner_alg, и происходит проверка удовлетворения выходной последовательности условию установки. В этом методе происходит отрисовка графика с помощью метода вспомогательной функции plot из библиотеки matplotlib.pyplot. Метод plot принимает на вход массив индексов и массив значений для создания графика. Метод process_alg возвращает эмпирическую моду - массив объектов типа Pair().

Метод full_alg реализует нахождение всех эмпирических мод для данной на вход последовательности. В этом методе происходит считывание данных из входного файла, отрисовка графика исходного сигнала, а так же вызывается метод process_alg пока входная последовательность не монотонна.

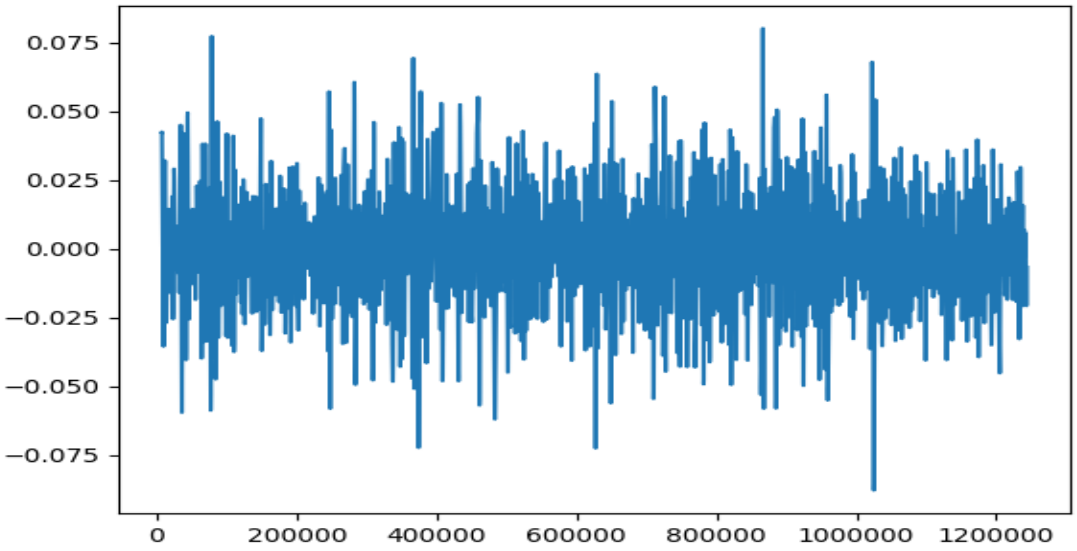
3. Полученные результаты

Для тестирования алгоритма были взяты данные фотоплетизмограммы (ФПГ) левого уха. Данные сняты с частотой 250 Гц. Коэффициент для условия остановки 0.25. Ниже представлены полученные графики.

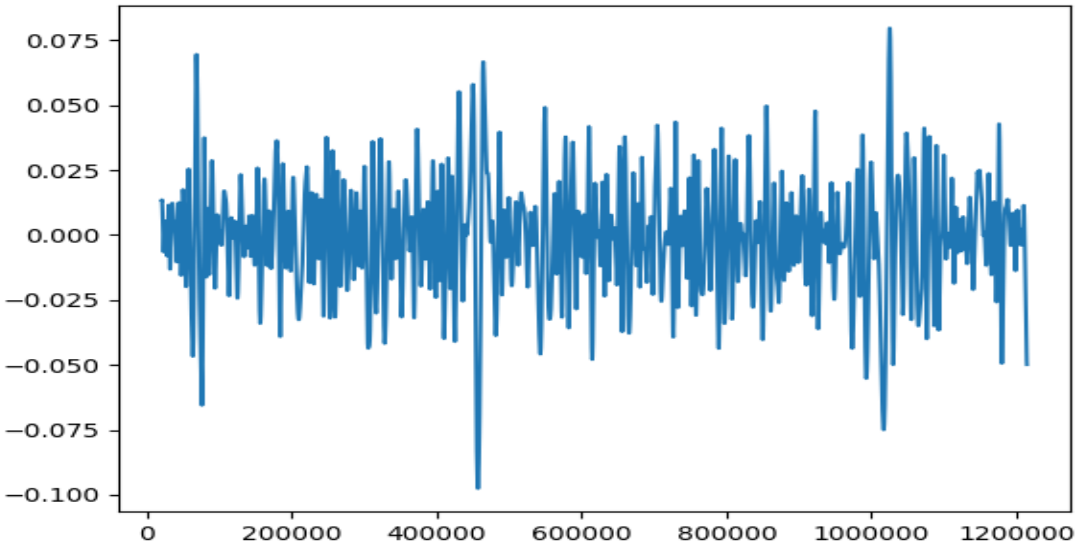
График исходных данные:



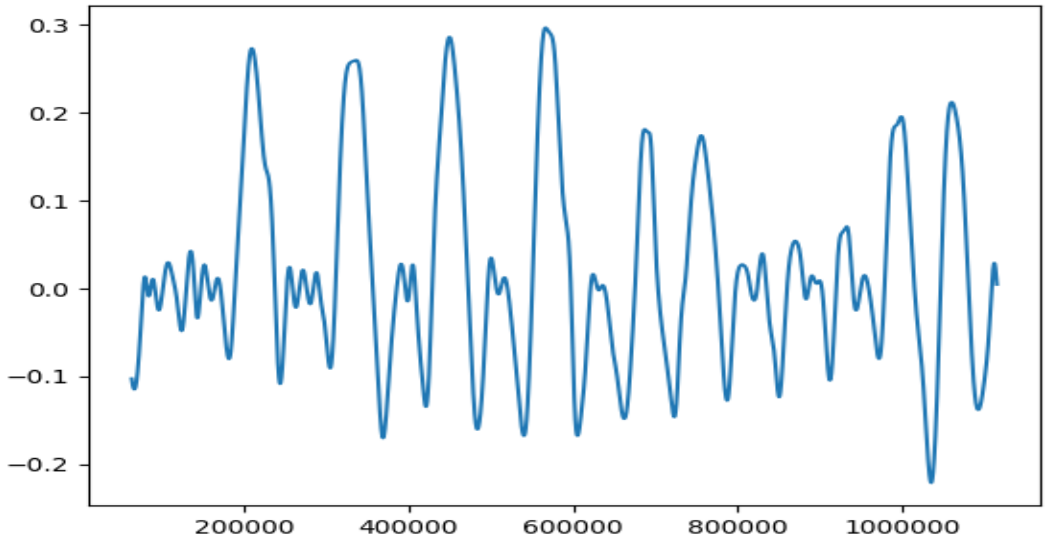
Эмпирическая мода в первом приближении:



Эмпирическая мода во втором приближении:



Эмпирическая мода в третьем приближении:



Далее последовательность стала монотонной и алгоритм остановился.

Так же были проведены тесты для сигналов снятых с левого и правого плеча человека (реограмма левого и правого плеча). В каждом из случаев наблюдается значительное упрощение исходного сигнала и исчезновение шумов.

Заключение

В ходе выполнения курсовой работы были рассмотрены различные методы декомпозиции данных: преобразование Фурье, спектрограмма, вейвлет-преобразование. Разобран метод эмпирической декомпозиции данных и реализован соответствующий алгоритм. В качестве внутренней задачи был рассмотрен алгоритм интерполяции кубическими сплайнами. Были проведены тесты на реальных данных, полученных в ходе медицинского обследования человека.

Основные преимущества метода эмпирической модовой декомпозиции перед другими рассмотренными:

1. Метод зависит исключительно от данных входного сигнала и использует базис преобразования, определяемый этим сигналом;
2. Метод подходит для нестационарных сигналов;
3. Метод относительно легко реализуем.

Метод удобен и может применяться во многих областях, например в медицине и обработке звуковых сигналов, но стоит помнить, что в ходе работы алгоритма получаются синтетические компоненты, которые помогают упростить процесс анализа сигнала, но их нельзя считать точными составляющими исходного процесса.

Список литературы

1. Статья «The empirical mode decomposition and the Hilbert spectrum for nonlinear and non-stationary time series analysis» by Norden E. Huang, Zheng Shen, Steven R. Long… от 3.06.1996.
2. Статья «A confidence limit for the empirical mode decomposition and Hilbert spectral analysis» by Norden E. Huang, Steven R. Long, Samuel S. P. Shen… от 4.09.2003.
3. «Знакомство с методом эмпирической декомпозиции мод», [Электронный ресурс]. URL: https://www.mql5.com/ru/articles/439 (дата обращения 17.02.2017)
4. Сафиулин Николай Тахирович, «Разработка методики анализа временных рядов с помощью преобразования Хуанга-Гильберта», ФГАОУ ВО «УрФУ имени первого Президента России Б.Н. Ельцина», 2015 г.
5. Е.А.Волков, «Численные методы», Москва, “Наука”, Главная редакция физико-математической литературы, 1987 г.
6. «Интерполяция кубическими сплайнами», [Электронный ресурс]. URL: http://su0.ru/Nvsg (дата обращения 17.02.2017)
7. «Создание графического интерфейса. Tkinter. Создание окна приложения», [Электронный ресурс]. URL: https://metanit.com/python/tutorial/9.1.php (дата обращения 10.08.2017)
8. «Международный научно-исследовательский журнал», [Электронный ресурс]. URL: https://research-journal.org/technical/ispolzovanie-preobrazovaniya-gilberta-xuanga-dlya-formirovaniya-modelej-fonem-russkogo-yazyka-v-zadache-raspoznavaniya-rechi/ (дата обращения 20.06.2017)

Приложение А

```
# -*- encoding: utf-8 -*-
import sys
sys.path.append("D:\\Programs\\Python2.7\\Lib\\site-packages\\alglib")
import xalglib as xal
from Tkinter import *
import tkFileDialog
import matplotlib.pyplot as ptl

class Pair:
    __index = -1
    __value = -1

    def setParams(self, index, value):
        self.index = index
        self.value = value

    def getIndex(self):
        return self.index

    def getValue(self):
        return self.value

    def toString(self):
        return str(self.index) + ':' + str(self.value)

filename = ""
imagenumb = 0
firstclickflag = True

def parse_data_from_file(file_name):
    file = open(file_name, 'r')
    list = []
    ind = 1
    for line in file:
        tmp = Pair()
        tmp.setParams(ind * 250, (float)(line))
        list.append(tmp)
        ind += 1
    file.close()
    return list

def parse_mid_result_file(file_name):
    file = open(file_name, 'r')
    list = []
    for line in file:
        words = line.split(":")
        tmp = Pair()
        tmp.setParams(float(words[0]), float(words[1]))
        list.append(tmp)
```

```
    file.close()
    return list

def find_top_extremum(list):
    t_max = []
    for i in range(1, len(list) - 2):
        if   list[i].getValue()   >   list[i   -   1].getValue()   and
list[i].getValue() >= list[i + 1].getValue():
            t_max.append(list[i])
    return t_max

def find_bot_extremum(list):
    t_min_1 = []
    for i in range(1, len(list) - 2):
        if   list[i].getValue()   <=   list[i   -   1].getValue()   and
list[i].getValue() < list[i + 1].getValue():
            t_min_1.append(list[i])
    return t_min_1

def find_interpolant(listExtremum):
    indexes = []
    values = []
    for elem in listExtremum:
        indexes.append(elem.getIndex())
        values.append(elem.getValue())
    ind = indexes[0]
    newIndexes = []
    while ind < indexes[indexes.__len__() - 1]:
        newIndexes.append(ind)
        ind += 50
    newValues = xal.spline1dconvcubic(indexes, values, newIndexes)
    returnArray = []
    for i in xrange(len(newIndexes)):
        tmp = Pair()
        tmp.setParams(newIndexes[i], newValues[i])
        returnArray.append(tmp)
        i += 1
    return returnArray

def find_index(index, array):
    i = 0
    while index != array[i].getIndex() and i < (array.__len__() - 1):
        i += 1
    return i

def find_middle_interpolation_values(topExtremum, botExtremum):
    newTopValues = find_interpolant(topExtremum)
    newBotValues = find_interpolant(botExtremum)
    middleDots = []
    if   newTopValues[0].getIndex()   >   newBotValues[0].getIndex()   and
newTopValues[-1].getIndex() > newBotValues[-1].getIndex():
```

```
        botIndex = find_index(newTopValues[0].getIndex(), newBotValues)
        i = 0
        while botIndex < newBotValues.__len__():
            tmp = Pair()
            tmp.setParams(newTopValues[i].getIndex(),
(newTopValues[i].getValue() + newBotValues[botIndex].getValue()) / 2)
            middleDots.append(tmp)
            i += 5
            botIndex += 5
    elif  newTopValues[0].getIndex()  >  newBotValues[0].getIndex()  and
newTopValues[-1].getIndex() < newBotValues[-1].getIndex():
        botIndex = find_index(newTopValues[0].getIndex(), newBotValues)
        i = 0
        while i < newTopValues.__len__():
            tmp = Pair()
            tmp.setParams(newTopValues[i].getIndex(),
(newTopValues[i].getValue() + newBotValues[botIndex].getValue()) / 2)
            middleDots.append(tmp)
            i += 5
            botIndex += 5
    elif  newTopValues[0].getIndex()  <  newBotValues[0].getIndex()  and
newTopValues[-1].getIndex() > newBotValues[-1].getIndex():
        topIndex = find_index(newBotValues[0].getIndex(), newTopValues)
        i = 0
        while i < newBotValues.__len__():
            tmp = Pair()
            tmp.setParams(newBotValues[i].getIndex(),
(newTopValues[topIndex].getValue() + newBotValues[i].getValue()) / 2)
            middleDots.append(tmp)
            i += 5
            topIndex += 5
    elif  newTopValues[0].getIndex()  <  newBotValues[0].getIndex()  and
newTopValues[-1].getIndex() < newBotValues[-1].getIndex():
        topIndex = find_index(newBotValues[0].getIndex(), newTopValues)
        i = 0
        while topIndex < newTopValues.__len__():
            tmp = Pair()
            tmp.setParams(newBotValues[i].getIndex(),
(newTopValues[topIndex].getValue() + newBotValues[i].getValue()) / 2)
            i += 5
            topIndex += 5
    return middleDots


def find_result(middleDots, prevResults):
    midIndex = 0
    prevIndex = 0
    result = []
    while midIndex < middleDots.__len__():
        if               middleDots[midIndex].getIndex()              !=
prevResults[prevIndex].getIndex():
```

```
            prevIndex += 1
        elif              middleDots[midIndex].getIndex()              ==
prevResults[prevIndex].getIndex():
            tmp = Pair()
            tmp.setParams(middleDots[midIndex].getIndex(),
                          prevResults[prevIndex].getValue()             -
middleDots[midIndex].getValue())
            result.append(tmp)
            prevIndex += 1
            midIndex += 1
    return result

def is_condition_met(currentSignal, prevSignal, constantToCompare):
    currentSignalIndex = 0
    prevSignalIndex = 0
    chisl = 0
    znam = 0
    while currentSignalIndex < currentSignal.__len__():
        if        currentSignal[currentSignalIndex].getIndex()        !=
prevSignal[prevSignalIndex].getIndex():
            prevSignalIndex += 1
        else:
            chisl    +=    (prevSignal[prevSignalIndex].getValue()    -
currentSignal[currentSignalIndex].getValue()) ** 2
            currentSignalIndex += 1
    for elem in prevSignal:
        znam += elem.getValue() ** 2
    if constantToCompare > (chisl / znam):
        return True
    else:
        return False

def is_monotonous(signal):
    return     find_top_extremum(signal).__len__()     ==     0     and
find_bot_extremum(signal).__len__() == 0

def choose_file():
    global filename
    filename = tkFileDialog.askopenfilename()

def one_step():
    global firstclickflag
    global filename
    global imagenumb
    if firstclickflag:
        firstclickflag = False
        data = parse_data_from_file(filename)
        indexlist = []
        valuelist = []
        i = 0
        while i < data.__len__()-2:
```

```
            indexlist.append(data[i].getIndex())
            valuelist.append(data[i].getValue())
            i += 1
        ptl.plot(indexlist, valuelist)

ptl.savefig('D:\ermolaxe\Programming\com.ermolaxe.courseproject\image\ste
palgorithm\\startdata.png')
        file                                                              =
open('D:\ermolaxe\Programming\com.ermolaxe.courseproject\\resources\middl
eresult.txt', 'w')
        for i in data:
            file.write(str(i.toString()+'\n'))
        file.close()
        filename                                                          =
'D:\ermolaxe\Programming\com.ermolaxe.courseproject\\resources\middleresu
lt.txt'
        ptl.show()
    else:
        data = parse_mid_result_file(filename)
        epsilon = float(coeff.get())
        notFinalResult = inner_alg(data)
        isConditionMet = is_condition_met(notFinalResult, data, epsilon)
        newNotFinalResult = notFinalResult
        while not isConditionMet:
            newNotFinalResult = inner_alg(notFinalResult)
            isConditionMet     =     is_condition_met(newNotFinalResult,
notFinalResult, epsilon)
            notFinalResult = newNotFinalResult
        indexlist = []
        valuelist = []
        i = 0
        while i < newNotFinalResult.__len__() - 2:
            indexlist.append(newNotFinalResult[i].getIndex())
            valuelist.append(newNotFinalResult[i].getValue())
            i += 1
        ptl.plot(indexlist, valuelist)

ptl.savefig('D:\ermolaxe\Programming\com.ermolaxe.courseproject\image\ste
palgorithm\\moda' + str(imagenumb) + '.png')
        imagenumb += 1
        if not is_monotonous(newNotFinalResult):
            newSignal = find_result(newNotFinalResult, data)
            file                                                          =
open('D:\ermolaxe\Programming\com.ermolaxe.courseproject\\resources\middl
eresult.txt', 'w')
            for i in newSignal:
                file.write(str(i.toString() + '\n'))
            file.close()
        ptl.show()
        ptl.clf()
```

```
def full_alg():
    data = parse_data_from_file(filename)
    epsilon = float(coeff.get())
    indexlist = []
    valuelist = []
    i = 0
    while i < data.__len__()-2:
        indexlist.append(data[i].getIndex())
        valuelist.append(data[i].getValue())
        i += 1
    ptl.plot(indexlist, valuelist)

ptl.savefig('D:\ermolaxe\Programming\com.ermolaxe.courseproject\image\\fu
llalgorithm\\startdata.png')
    ptl.show()
    ptl.clf()
    result = process_alg(data, epsilon)
    isMonotonous = is_monotonous(result)
    while not isMonotonous:
        result = process_alg(result, epsilon)
        isMonotonous = is_monotonous(result)
    return result

def process_alg(data, epsilon):
    global imagenumb
    notFinalResult = inner_alg(data)
    isConditionMet = is_condition_met(notFinalResult, data, epsilon)
    newNotFinalResult = notFinalResult
    while not isConditionMet:
        newNotFinalResult = inner_alg(notFinalResult)
        isConditionMet       =       is_condition_met(newNotFinalResult,
notFinalResult, epsilon)
        notFinalResult = newNotFinalResult
    indexlist = []
    valuelist = []
    i = 0
    while i < newNotFinalResult.__len__() - 2:
        indexlist.append(newNotFinalResult[i].getIndex())
        valuelist.append(newNotFinalResult[i].getValue())
        i += 1
    ptl.plot(indexlist, valuelist)

ptl.savefig('D:\ermolaxe\Programming\com.ermolaxe.courseproject\image\\fu
llalgorithm\\moda'+str(imagenumb)+'.png')
    imagenumb += 1
    ptl.clf()
    result = find_result(newNotFinalResult, data)
    return result

def inner_alg(data):
    topExtremum = find_top_extremum(data)
```

```
    botExtremum = find_bot_extremum(data)
    if topExtremum.__len__() >= 2 and botExtremum.__len__() >= 2:
        middleDots    =    find_middle_interpolation_values(topExtremum,
botExtremum)
        notFinalResult = find_result(middleDots, data)
        return notFinalResult
    else:
        return data

# user interface
root = Tk()
root.title("Эмпирическая декомпозиция мод")
root.geometry("460x100")
root.config(background="#D3F8B0")
choose_file_btn = Button(root, text='Выбрать файл', background="#9DD06D",
foreground="#27292B", padx=3, pady=3,
                         command=choose_file).place(x=20,           y=10,
width=100, height=50)
one_step_btn    =    Button(root,    text='Один    шаг    алгоритма',
background="#9DD06D", foreground="#27292B", padx=3, pady=3,
                       command=one_step).place(x=130,   y=10,   width=140,
height=50)
full_alg_btn    =    Button(root,    text='Выполнить    весь    алгоритм',
background="#9DD06D", foreground="#27292B", padx=3, pady=3,
                       command=full_alg).place(x=280,   y=10,   width=160,
height=50)
radioNumber = IntVar()
radioNumber.set(1)
lab   =   Label(root,   bg="#D3F8B0",   text='Коэффициент   для   сравнения
нормированного квадрата разностей')
lab.place(x=10, y=72)
coeff = Entry(root, width=10, bd=2)
coeff.place(x=385, y=72)
     root.mainloop()
```