Лабораторная работа №4

Построение вейвлетов Добеши и их графиков

Вариант 4 - 1
Выполнил Казачинский Глеб, 3 курс 6 группа

Постановка задачи

Дан тип ортономированного вейвлета Добеши с нулевыми моментами. Используя процедуру спектральной факторизации, вычислить коэффициенты двухмасштабного соотношения (ДМС) $\{h_n\}$. Построить график построенной масштабирующей функции и вейвлета, используя указанный в варианте способ.

Для осуществления спектральной факторизации разрешается использовать ПО для символьных вычислений (Mathematica, Maple, Matlab и т. п.)

Содержание отчета:

- Подробное описание процесса получения коэффициентов ДМС с приведением кода на используемом для этого языке.
- Сравнение полученных коэффициентов с приведенными в [Добеши].
- В силу неединственности спектральной факторизации также приветствуется получение одного-двух других вариантов ДМС (а также графиков соответствующих МФ и вейвлета), отличных от классических.
- Графики масштабирующей функции и вейвлета.
- Описание алгоритма, с помощью которого были построены графики.
- Код всех программ.

- Подробное описание процесса получения коэффициентов ДМС с приведением кода на используемом для этого языке.

Приведём код функции *get_daubechies_coef(N,* a_N*),* принимающая на вход N и a_N(a_N сложно вычислять для любого N в программе, приходится считать a_N отдельно, в вольфраме) и отдающая на выход коэффициенты ДМС(весь код программы в конце отчёта) :

```python
def get_daubechies_coef(N, a_N):
    Q_ = Q(N)
    U_ = U(N, Q_)
    U_roots = U_.roots()

    N_1, N_2, N_3 = get_N1_N2_N3(U_roots)

    z_k_1 = z_k(U_roots, N_1, 'real')
    z_k_2 = z_k(U_roots, N_2, 'real')
    z_k_3 = z_k(U_roots, N_3, 'complex')

    B_1_ = B_1(N_1, z_k_1)
    B_2_ = B_2(N_2, z_k_2)
    B_3_ = B_3(N_3, z_k_3)

    B_ = B(a_N, B_1_, B_2_, B_3_)

    M_0_sqrt_2 = M_0(N, B_) * np.sqrt(2)

    return M_0_sqrt_2.coef
```

- Сравнение полученных коэффициентов с приведенными в [Добеши].

```
|daubechies_coef - daubechies_coef_true| = 3.638974036064114e-15
```

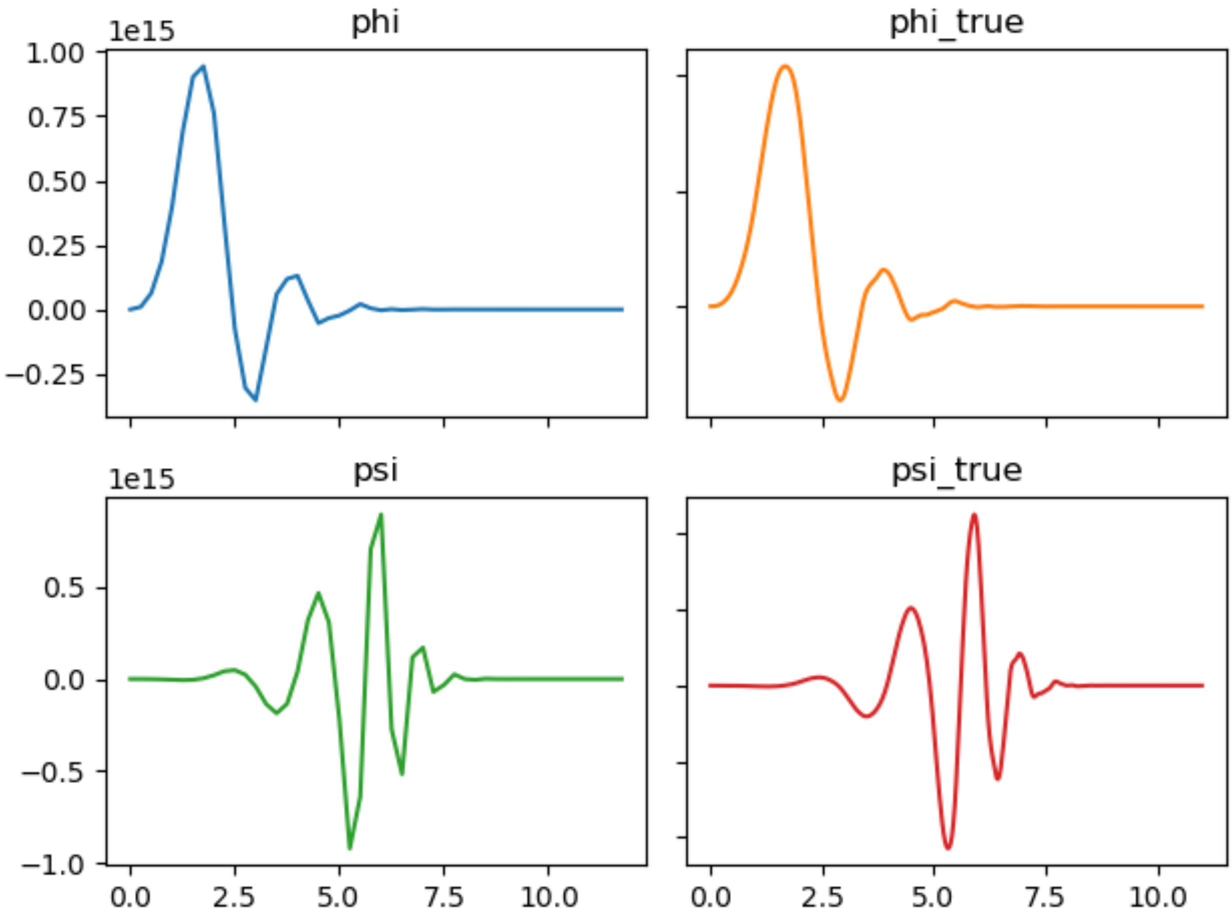
Полученные коэффициенты сравниваю с ***pywt.Wavelet('db6').dec_lo***
Получили неплохую невязку

- Графики масштабирующей функции и вейвлета.

Приведём код функции *draw_scaling_function_and_wavelet(h),* принимающая на вход h и отдающая на выход графики для φ и ψ используя запрограммированный алгоритм и ***pywt.Wavelet('db6')***(весь код программы в конце отчёта):

```
def draw_scaling_function_and_wavelet(h):
    left, right = 0, 12
    x = np.arange(left, right, 1 / 4)
    N = len(x)

    fi, fi_obj = get_scaling_function(h[::-1], x, N, left, right)

    psi = get_wavelet_function(h, x, fi_obj)

    wavelet = pywt.Wavelet('db6')
    phi_true, psi_true, x_true = wavelet.wavefun(level=5)

    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
    ax1.plot(x, fi)
    ax1.set_title('phi')
    ax2.plot(x_true, phi_true, 'tab:orange')
    ax2.set_title('phi_true')
    ax3.plot(x, psi, 'tab:green')
    ax3.set_title('psi')
    ax4.plot(x_true, psi_true, 'tab:red')
    ax4.set_title('psi_true')

    for ax in fig.get_axes():
        ax.label_outer()

    plt.show()
```

Получили графики:



- Описание алгоритма, с помощью которого были построены графики.

Запрограммированы следующие формулы:

$$\varphi = \sum_n h_n \varphi_{-1,n}$$

$$\psi = \sum_n q_n \varphi_{-1,n} \text{ ,где } q_n = \overline{h_{1-n}}(-1)^n$$

За начальное приближение для φ беру кусочно-непрерывную функцию, равную 1 на отрезке [0, 12], нулю в противном случае

- Код всех программ.

Код программы для получения коэффициентов ДМС:

```
import numpy as np
import scipy.special as scp
import numpy.polynomial.polynomial as poly
from numpy import linalg as LA
from wavelet_analysis.l4.draw_daubechies import draw_scaling_function_and_wavelet
import pywt as pywt


def Q(N):
    polynom_coefs = np.zeros(N)
    for k in range(N):
        polynom_coefs[k] = scp.binom(N - 1 + k, k)
    return poly.Polynomial(polynom_coefs)


def U(N, Q):
    result_poly = 0
    for i in range(N):
        result_poly += Q.coef[i] * poly.Polynomial([1 / 2, -1 / 2]) ** i
    return result_poly


def z_k(U_roots, N, type):
    ret_value = []
    for k in range(len(U_roots)):
        ret_value.append(U_roots[k] + np.sqrt(U_roots[k] ** 2 - 1))
        ret_value.append(U_roots[k] - np.sqrt(U_roots[k] ** 2 - 1))
    print(ret_value)
    count = 0
    z_k = []
    z_k_blacklist = []
    if type == 'complex':
        for i in range(len(ret_value)):
            if not np.isin(np.abs(z_k_blacklist), np.abs(ret_value[i])) and ret_value[i].real < 1 and ret_value[i].imag < 1:
                z_k.append(ret_value[i])
                z_k_blacklist.append(ret_value[i])
                count += 1
            if count == N:
                break
    if type == 'real':
        for i in range(len(ret_value)):
            if ret_value[i].imag == 0 and np.abs(ret_value[i]) <= 1:
```

```
                z_k.append(ret_value[i])
                count += 1
            if count == N:
                break
    return z_k


def B_1(N_1, z_k):
    r_k = z_k
    ret_value = poly.Polynomial([1])
    for k in range(1, N_1 + 1):
        ret_value *= (poly.Polynomial([- r_k[k - 1], 1])) / np.sqrt(np.abs(r_k[k - 1]))
    return ret_value


def B_2(N_2, z_k):
    cos_alpha_k = map(lambda x: x.real, z_k)
    ret_value = poly.Polynomial([1])
    for k in range(1, N_2 + 1):
        ret_value *= poly.Polynomial([1, - 2 * cos_alpha_k[k - 1], 1])
    return ret_value


def B_3(N_3, z_k):
    ret_value = poly.Polynomial([1])
    for k in range(1, N_3 + 1):
        ret_value *= (poly.Polynomial([np.abs(z_k[k - 1]) ** 2, - 2 * z_k[k - 1].real, 1]) / np.abs(z_k[k - 1]))
    return ret_value


def B(a_N, B_1=1, B_2=1, B_3=1):
    return np.sqrt(np.abs(a_N) / 2) * B_1 * B_2 * B_3


def M_0(N, B):
    return poly.Polynomial([1 / 2, 1 / 2]) ** N * B


def get_N1_N2_N3(U_roots):
    N_1, N_2, N_3 = 0, 0, 0
    for i in range(len(U_roots)):
        if isinstance(U_roots[i], complex) and U_roots[i].imag != 0:
            N_3 += 1
        else:
            if np.abs(U_roots[i] >= 1):
                N_1 += 1
            else:
                N_2 += 1
```

```
    return N_1, N_2, N_3 // 2


def get_Q_special(Q):
    ret_coef = [Q.coef[0]]
    for i in range(1, len(Q.coef)):
        ret_coef.append(Q.coef[i] / (2 ** i))
    print(ret_coef)


def get_daubechies_coef(N, a_N):
    Q_ = Q(N)
    U_ = U(N, Q_)
    U_roots = U_.roots()

    N_1, N_2, N_3 = get_N1_N2_N3(U_roots)

    z_k_1 = z_k(U_roots, N_1, 'real')
    z_k_2 = z_k(U_roots, N_2, 'real')
    z_k_3 = z_k(U_roots, N_3, 'complex')

    B_1_ = B_1(N_1, z_k_1)
    B_2_ = B_2(N_2, z_k_2)
    B_3_ = B_3(N_3, z_k_3)

    B_ = B(a_N, B_1_, B_2_, B_3_)

    M_0_sqrt_2 = M_0(N, B_) * np.sqrt(2)

    return M_0_sqrt_2.coef


N = 6
a_N = - 63 / 128

daubechies_coef = get_daubechies_coef(N, a_N)

wavelet = pywt.Wavelet('db6')
print(f'|daubechies_coef - daubechies_coef_true| = {LA.norm(daubechies_coef - wavelet.dec_lo)}')
draw_scaling_function_and_wavelet(daubechies_coef)
```

Код программы для построения графиков:

```python
import numpy as np
from matplotlib import pyplot as plt
import pywt as pywt


def phi_0(x, left, right):
    if x < left or x > right:
        return 0
    return 1


def draw_scaling_function_and_wavelet(h):
    left, right = 0, 12
    x = np.arange(left, right, 1 / 4)
    N = len(x)

    fi, fi_obj = get_scaling_function(h[::-1], x, N, left, right)

    psi = get_wavelet_function(h, x, fi_obj)

    wavelet = pywt.Wavelet('db6')
    phi_true, psi_true, x_true = wavelet.wavefun(level=5)

    fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2)
    ax1.plot(x, fi)
    ax1.set_title('phi')
    ax2.plot(x_true, phi_true, 'tab:orange')
    ax2.set_title('phi_true')
    ax3.plot(x, psi, 'tab:green')
    ax3.set_title('psi')
    ax4.plot(x_true, psi_true, 'tab:red')
    ax4.set_title('psi_true')

    for ax in fig.get_axes():
        ax.label_outer()

    plt.show()

```

```python
def get_scaling_function(h, x, N, left, right):
    fi_1, fi_1_object = np.zeros(len(x), dtype=float), {}

    for xi in range(len(x)):
        for k in range(len(h)):
            fi_1[xi] += np.sqrt(2) * h[k] * phi_0(2 * x[xi] - k, left, right)
        fi_1_object[x[xi]] = fi_1[xi]

    fi_2 = np.zeros(len(x), dtype=float)

    fi_2_object = {}
    for m in range(1, N):
        fi_2_object = {}
        for x_i, i in zip(x, range(len(x))):
            for k in range(len(h)):
                if 2 * x_i - k in x:
                    fi_2[i] += np.sqrt(2) * h[k] * fi_1_object[2 * x_i - k]
            fi_2_object[x_i] = fi_2[i]
        fi_1_object = fi_2_object

    return fi_2, fi_2_object


def get_wavelet_function(h, x, fi_obj):
    q = np.zeros(len(h))
    for k in range(len(h)):
        q[k] = ((-1) ** k) * h[k]

    psi = np.zeros(len(x))
    for x_i, i in zip(x, range(len(x))):
        for k in range(len(h)):
            if 2 * x_i - k in x:
                psi[i] += np.sqrt(2) * q[k] * fi_obj[2 * x_i - k]
    return psi

```