Федеральное государственное бюджетное образовательное
учреждение высшего профессионального образования Московский
государственный технический университет имени Н.Э. Баумана

Лабораторная работа № 4
«Исследование стохастической фильтрации сигналов как задачи двухкритериальной опитимизации с использованием методов прямого пассивного поиска»
по курсу
«Теория игр»

Студент группы ИУ9-31М Белогуров А.А.

Преподаватель Басараб М.А.

Москва, 2019

Содержание

1 Цель работы **3**

2 Постановка задачи **4**

3 Практическая реализация **6**

4 Результаты **11**

1 Цель работы

Изучить основные принципы многокритериальной оптимизации в комбинации с методами случайного и прямого пассивного поиска применительно к задаче фильтрации дискретного сигнала методом взвешенного скользящего среднего.

2 Постановка задачи

Вариант 2.

На интервале $[x_{min}, x_{max}]$ задан сигнал $f_k = f(x_k)$, где дискретная последовательность отсчетов $x_k = x_{min} + k(x_{max} - x_{min})/K, k = 0, ..., K, K$ - количество отсчетов. На сигнал наложен дискретный равномерный шум $\sigma = (\sigma_0, ..., \sigma_K)$ с нулевым средним и амплитудой, равномерно распределенной на интервале $[-a, a] : \tilde{f}_k = f_k + \sigma_k, \sigma_k = rnd(-a, a)$. В зависимости от варианта работы необходимо осуществить фильтрацию сигнала $\tilde{f}_k$ одним из методов взвешенного скользящего среднего - среднее геометрическое:

$$\overline{f_k}(\alpha) = \prod_{j=k-M}^{k+M} \tilde{f}_j^{\alpha_{j+M+1-k}}$$

При расчете критериев зашумленности и близости использовать "Манхеттенкскую"метрику:

$$\omega = \sum_{k=1}^{K} |\overline{f}_k - \overline{f}_{k-1}|, \quad \delta = \frac{1}{K} \sum_{k=0}^{K} |\overline{f}_k - \tilde{f}_k|$$

Линейная свертка критериев:

$$J = \lambda\omega + (1 - \lambda)\delta \to \min_{\alpha}, \quad \lambda \in [0, 1]$$

При расчете расстояния до идеальной точки использовать "Манхеттенкскую"метрику:

$$dist = |\omega| + |\delta|$$

Конечная цель работы заключается в нахождении оптимального веса λ^* (методом прямого пассивного поиска на сетке λ_l), при котором минимизируется расстояние от приближенно найденного оптимального значения интегрального критерия $J^*(\omega^*, \delta^*)$ до идеальной-точки $\hat{J}(\hat{\omega}, \hat{\delta}) = \hat{J}(0, 0) = 0$:

$$dist(J^*, \hat{J}) \to \min_{\lambda}.$$

3 Практическая реализация

Main.py

```
import matplotlib.pyplot as plt
import numpy as np


def get_default_signal():
    # Исходный сигнал
    K = 100
    k = np.arange(0, K)
    x_min = 0
    x_max = np.pi
    x = x_min + k * (x_max - x_min) / K
    f = np.sin(x) + 0.5

    return K, k, x_min, x_max, x, f


def get_noise_amplitude():
    # Амплитуда равномерного шума
    a = 0.5 / 2
    return a


def get_convolution_weight_discretization():
    # Дискретизация веса свертки
    L = 10
    l = np.arange(0, L)
    lambda_val = l / L
    return lambda_val


def get_probability_extremum():
    # Вероятность попадания в окрестность экстремума
    P = 0.95
    return P


def get_uncertainty_interval():
```

```
    # Интервал неопределенности
    epsilon = 0.01
    return epsilon


def get_sliding_window_size():
    # Размер скользящего окна
    r_3 = 3
    r_5 = 5
    return r_3, r_5


def cacl_f_weight_mean(r, f_with_sigma, alpha=None):
    # Считаем взвешенное скользящее среднее
    M = int((r - 1) / 2)
    if alpha is None:
        alpha = [1 / r] * r

    f_weight_mean = []
    for k in range(M, len(f_with_sigma) - M):
        f_temp = []
        for j in range(k - M, k + M + 1):
            f_temp.append(f_with_sigma[j] ** alpha[j + M - k])

        f_weight_mean.append(np.prod(f_temp))

    return f_weight_mean


def calc_noisy_criterion(f_with_sigma):
    # Считаем критерий зашумленности по "Манхеттонской" метрике
    omega = 0
    for k in range(1, len(f_with_sigma)):
        omega += abs(f_with_sigma[k] - f_with_sigma[k - 1])

    return omega


def calc_proximity_criterion(f_with_sigma, f_weight_mean):
    # Считаем критерий близости по "Манхеттонской" метрике
    delta = 0
    for k in range(1, len(f_weight_mean)):
```

```
        delta += abs(f_with_sigma[k] - f_weight_mean[k])

    return delta / len(f_with_sigma)


if __name__ == "__main__":
    np.set_printoptions(precision=2)

    K, k, x_min, x_max, x, f = get_default_signal()
    plt.grid()
    plt.plot(k, f, label='Исходный сигнал')
    plt.legend()
    plt.show()

    # Добавляем шум
    a = get_noise_amplitude()
    sigma = np.random.uniform(low=-a, high=a, size=(K,))
    f_with_sigma = f + sigma

    plt.grid()
    plt.plot(k, f, label='Исходный сигнал', alpha=0.3)
    plt.plot(k, f_with_sigma, label='Исходный сигнал плюс шум')
    plt.legend()
    plt.show()

    # Среднее геометрическое для окна r = 3 и r = 5
    r_3, r_5 = get_sliding_window_size()
    f_weight_mean_3 = cacl_f_weight_mean(r_3, f_with_sigma)
    f_weight_mean_5 = cacl_f_weight_mean(r_5, f_with_sigma)

    plt.grid()
    plt.plot(k, f, label='Исходный сигнал', alpha=0.3)
    plt.plot(k, f_with_sigma, label='Исходный сигнал плюс шум', alpha=0.3)
    plt.plot(k[1:-1], f_weight_mean_3, label='$\widetilde{f_k}$ с окном $r=3$')
    plt.plot(k[2:-2], f_weight_mean_5, label='$\widetilde{f_k}$ с окном $r=5$')
    plt.legend()
    plt.show()

    # Подбираем alpha
```

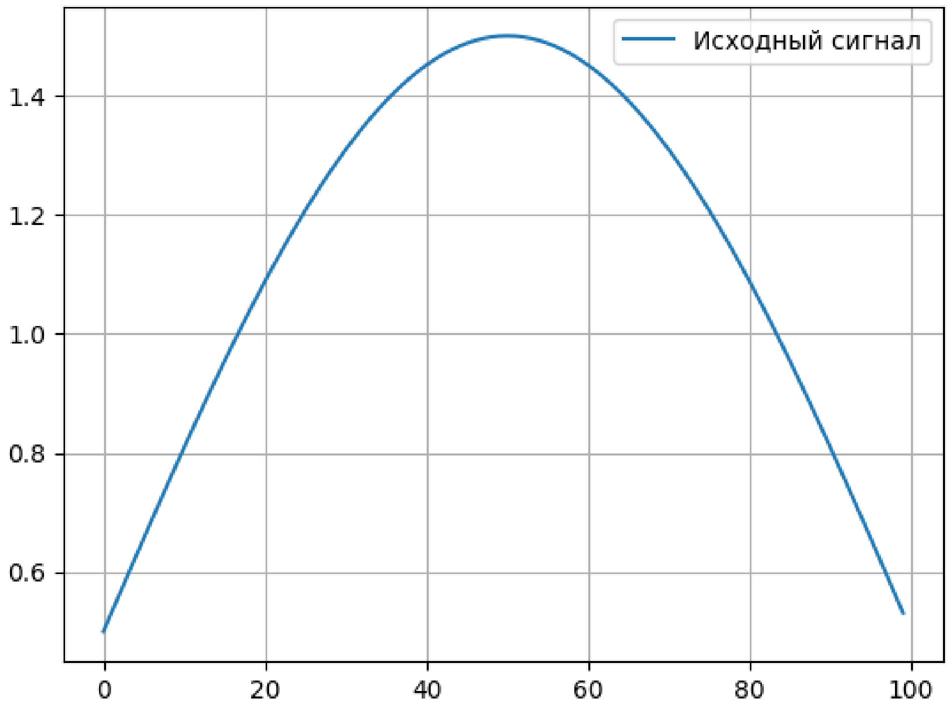
```
    J = []
    omegas = []
    deltas = []
    dists = []
    lambdas = get_convolution_weight_discretization()

    for lambda_val in lambdas:
        alpha_center = lambda_val
        alpha_prev_next = (1 - alpha_center) / 2
        alpha = np.around([alpha_prev_next, alpha_center, alpha_prev_next], decimals=2)

        f_weight_mean_3_alpha = cacl_f_weight_mean(r_3, f_weight_mean_3, alpha)
        omega = calc_noisy_criterion(f_weight_mean_3_alpha)
        delta = calc_proximity_criterion(f_with_sigma, f_weight_mean_3_alpha)
        dist = abs(omega) + abs(delta)

        J_local = lambda_val * delta + (1 - lambda_val) * delta

        J.append(J_local)
        omegas.append(omega)
        deltas.append(delta)
        dists.append(dist)

        plt.figure(figsize=(20, 3))
        plt.grid()
        plt.title(
            f'$\\alpha =$ {alpha}, $\omega = {omega:.3f}$, $\delta = {delta:.3f}$, dist = {dist:.3f}, J = {J_local:.3f}')
        plt.plot(k, f_with_sigma, label='Исходный сигнал плюс шум', alpha=0.3)
        plt.plot(k[2:-2], f_weight_mean_3_alpha, label='Взвешенный сигнал')
        plt.show()

    # Зависимость sigma, delta от lambda
    plt.grid()
    plt.title(f'Зависимость $(\omega, \delta)$ от $\lambda$')
    plt.plot(lambdas, omegas, label='$\omega$')
```

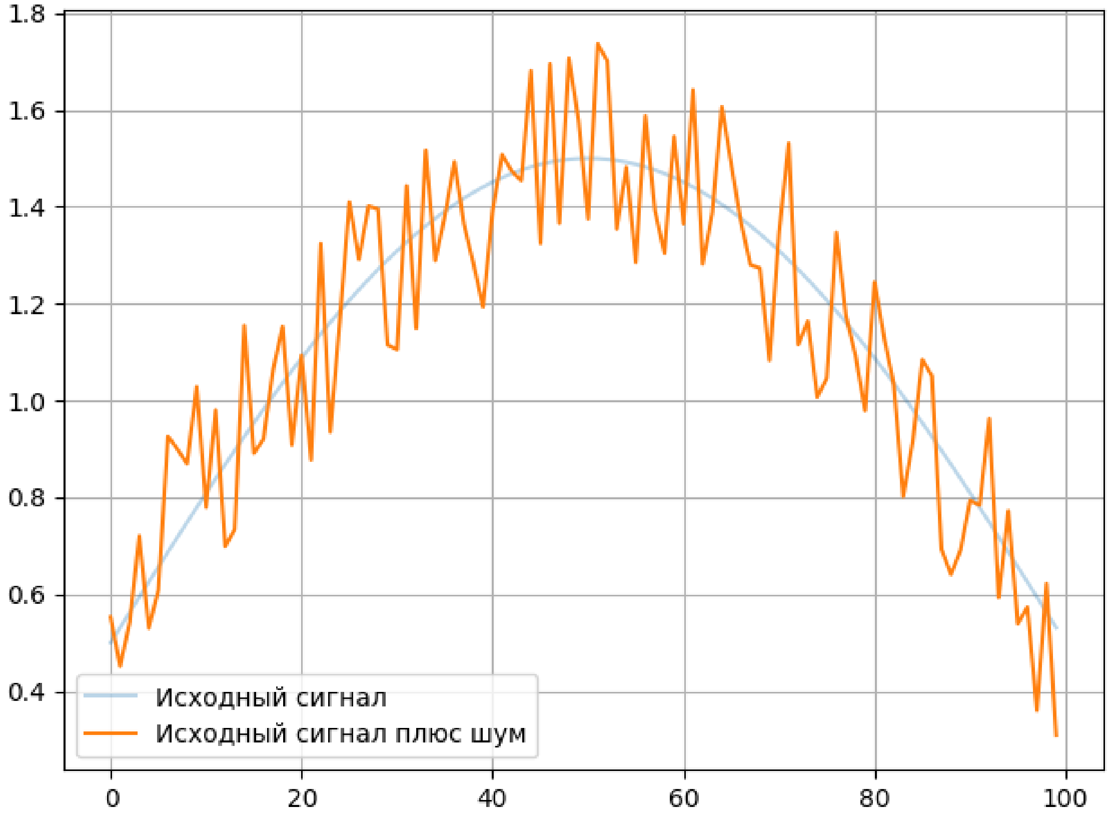
```
    plt.plot(lambdas, deltas, label='$\delta$')
    plt.legend()
    plt.ylabel('$(\omega, \delta)$')
    plt.xlabel('$(\lambda)$')
    plt.show()

    # Зависимость расстояния от lambda
    plt.grid()
    plt.title(f'Зависимость расстояния от $\lambda$')
    plt.plot(lambdas, dists)
    plt.ylabel('dists')
    plt.xlabel('$\lambda$')
    plt.show()


    P = get_probability_extremum()
    epsilon = get_uncertainty_interval()
    N = np.log(1 - P) / np.log(1 - (epsilon / (x_max - x_min)))

    print(f'Число испытаний = {N}')
    print(f'Минимальное значение расстояния = {min(np.around(dists, 2))}')
```

4 Результаты

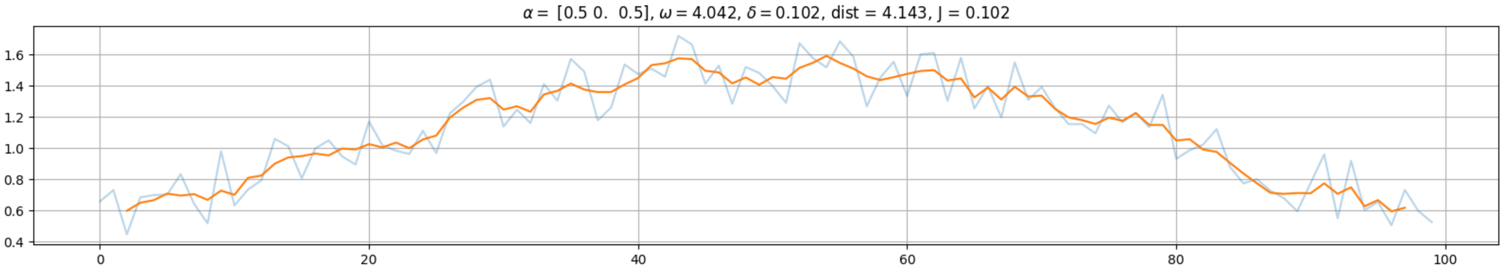
Процесс работы программы и результаты:

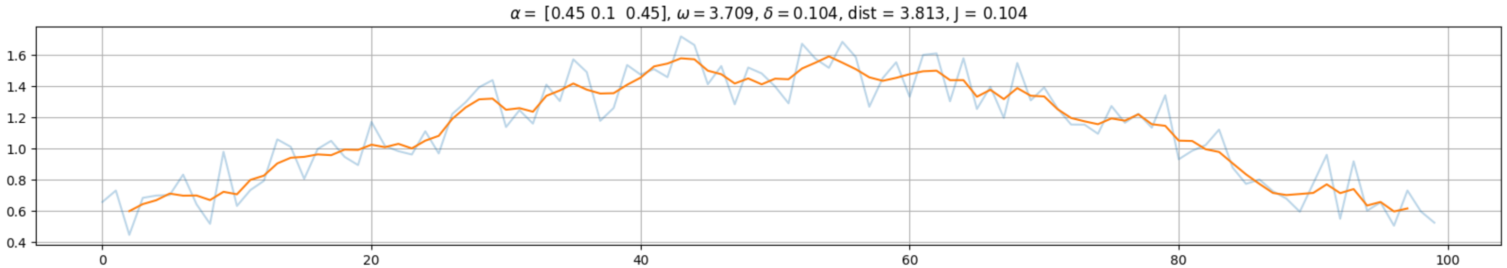
```
Число испытаний = 939.6383882252046
Минимальное значение расстояния = 3.435
```

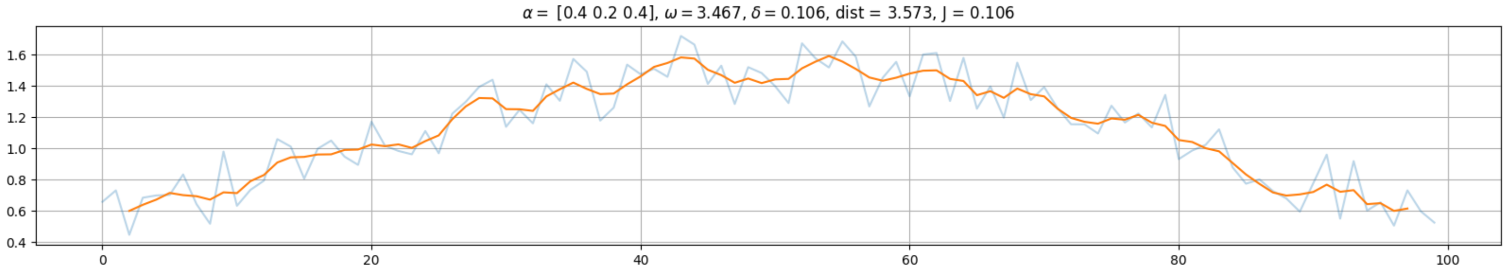


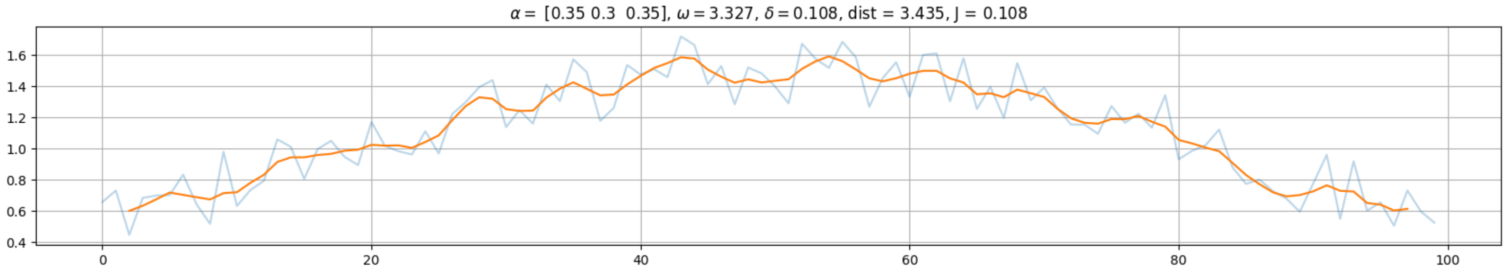

1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0
20
40
60
80
100
Исходный сигнал
Исходный сигнал плюс шум

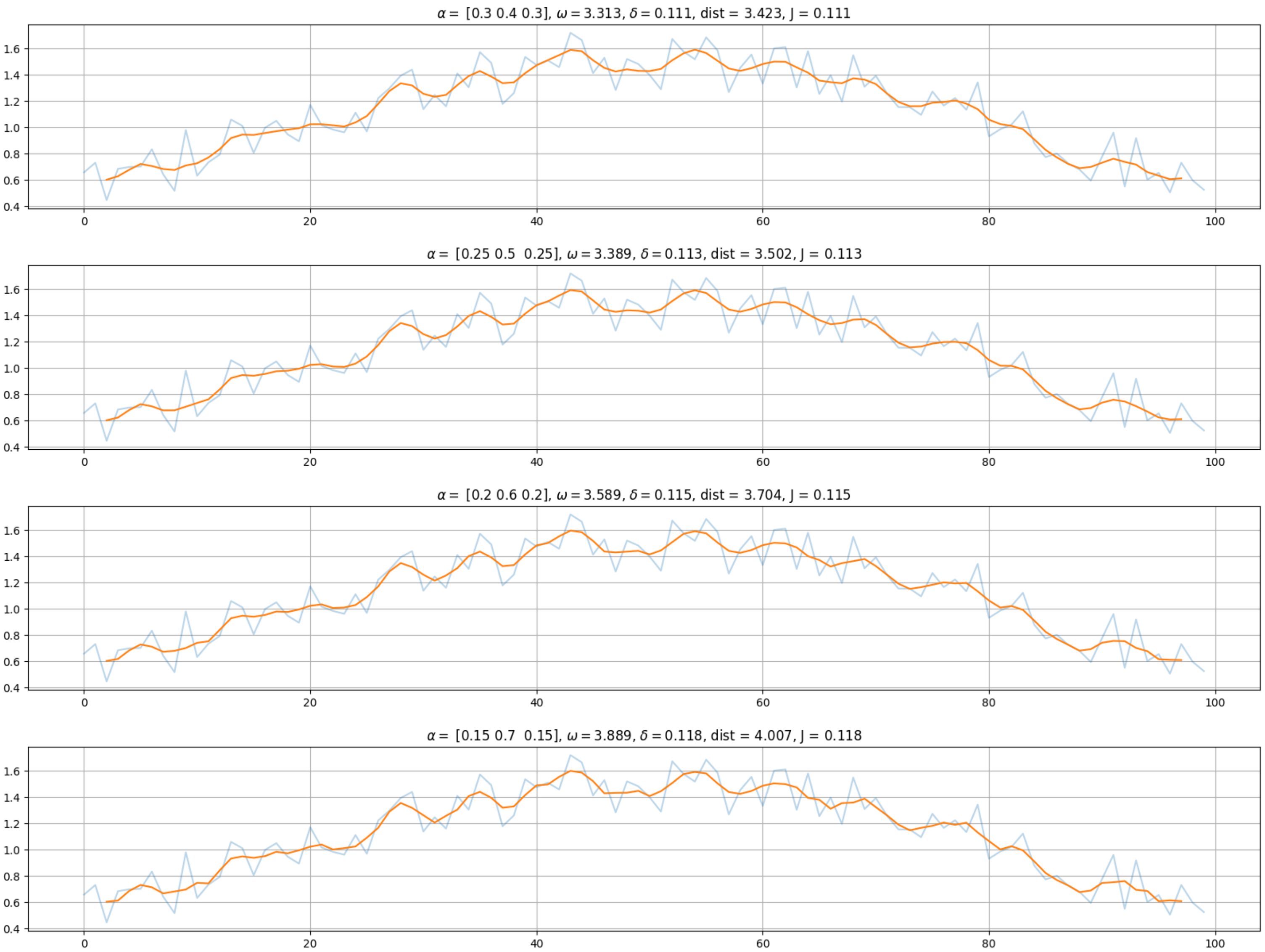
Исходный сигнал
Исходный сигнал плюс шум
$\widetilde{f_k}$ с окном $r = 3$
$\widetilde{f_k}$ с окном $r = 5$

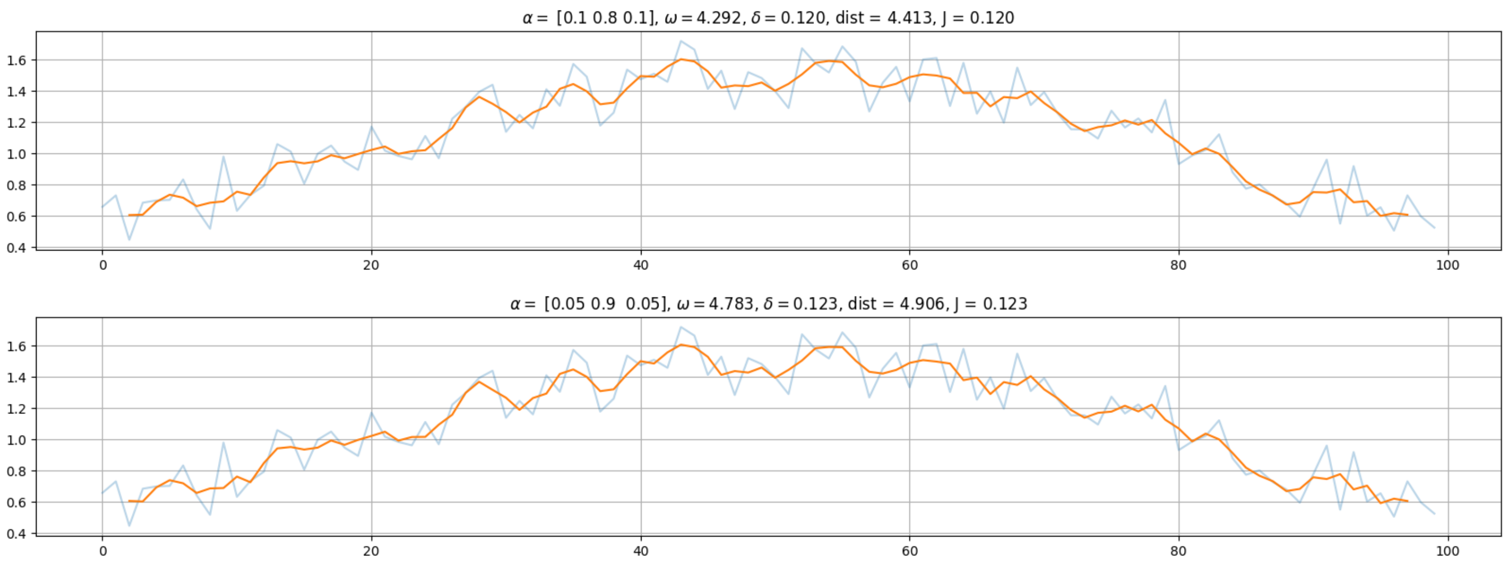

$\alpha =$ [0.5 0. 0.5], $\omega = 4.042$, $\delta = 0.102$, dist = 4.143, J = 0.102

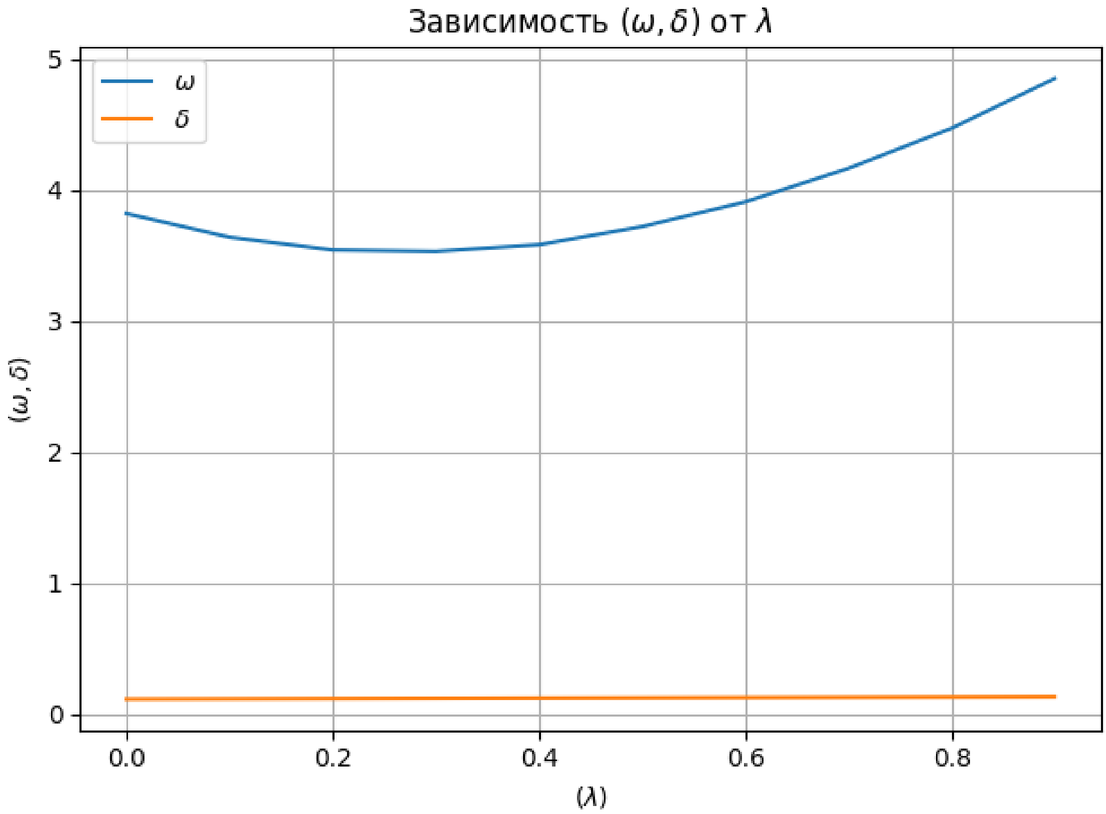

$\alpha =$ [0.45 0.1 0.45], $\omega = 3.709$, $\delta = 0.104$, dist = 3.813, J = 0.104

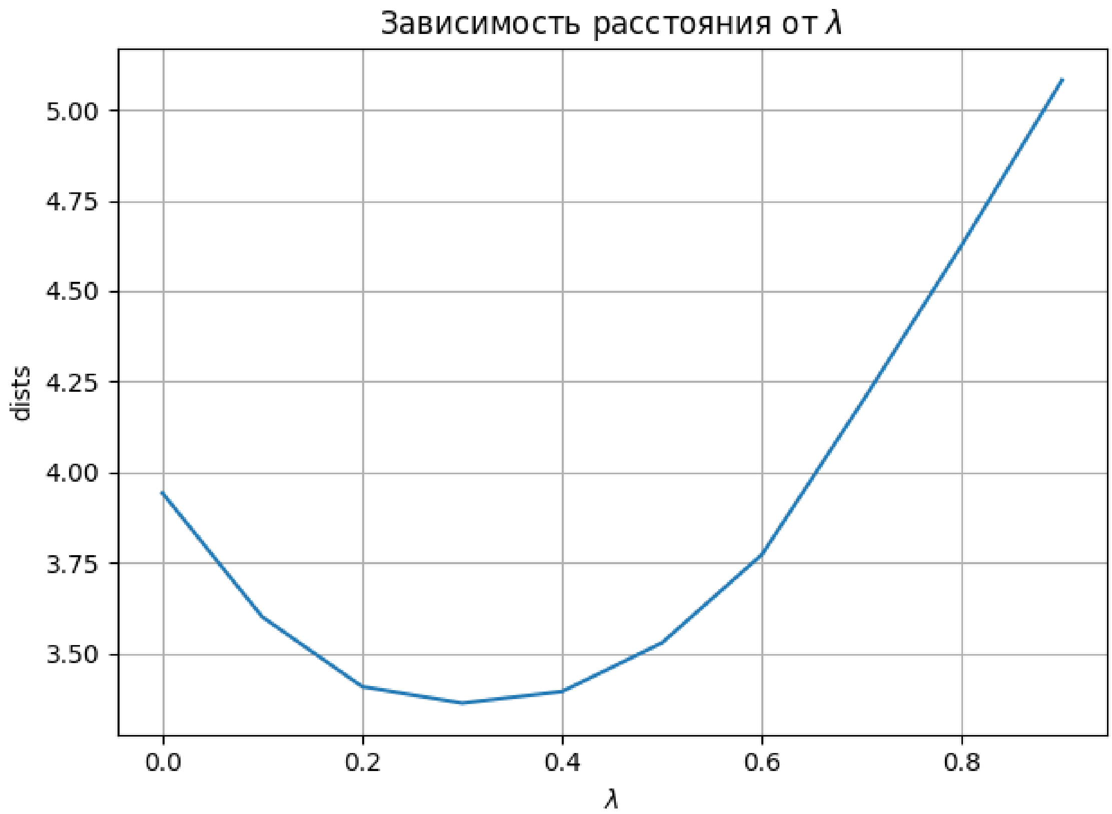

$\alpha =$ [0.4 0.2 0.4], $\omega = 3.467$, $\delta = 0.106$, dist = 3.573, J = 0.106


$\alpha =$ [0.35 0.3 0.35], $\omega = 3.327$, $\delta = 0.108$, dist = 3.435, J = 0.108


α = [0.3 0.4 0.3], ω = 3.313, δ = 0.111, dist = 3.423, J = 0.111
α = [0.25 0.5 0.25], ω = 3.389, δ = 0.113, dist = 3.502, J = 0.113
α = [0.2 0.6 0.2], ω = 3.589, δ = 0.115, dist = 3.704, J = 0.115
α = [0.15 0.7 0.15], ω = 3.889, δ = 0.118, dist = 4.007, J = 0.118


$\alpha =$ [0.1 0.8 0.1], $\omega = 4.292$, $\delta = 0.120$, dist = 4.413, J = 0.120
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
20
40
60
80
100
$\alpha =$ [0.05 0.9 0.05], $\omega = 4.783$, $\delta = 0.123$, dist = 4.906, J = 0.123
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
20
40
60
80
100


Зависимость (ω, δ) от λ
ω
δ
5
4
3
2
1
0
(ω, δ)
0.0
0.2
0.4
0.6
0.8
(λ)


Зависимость расстояния от λ
5.00
4.75
4.50
4.25
4.00
3.75
3.50
dists
0.0
0.2
0.4
0.6
0.8
λ