Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления
Кафедра: Информационная безопасность
(ИУ8)

ТЕОРИЯ СИСТЕМ И СИСТЕМНЫЙ АНАЛИЗ

Лабораторная работа №6 на тему:

«Исследование стохастической фильтрации сигналов как задачи многокритериальной оптимизации с использованием методов прямого пассивного поиска»

Вариант 7

Преподаватель:
Строганов И.С.

Студент:
Заботин Д.В.

Группа:
ИУ8-31

Москва 2021

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.

Постановка задачи

На интервале $[x_{min}, x_{max}]$ задан сигнал $f_k = f(x_k)$, где дискретная последовательность отсчетов $x_k = x_{min} + \frac{k(x_{max} - x_{min})}{K}$, k = 0, … K, K – количество отсчетов. На сигнал наложен дискретный шум $\sigma = (\sigma_0, \dots, \sigma_k)$ с нулевым средним амплитудой, равномерно распределенной на интервале [-a, a] $\widetilde{f_k} = f_k + \sigma_k$, σ_k = rand(-a, a). Необходимо осуществить фильтрацию сигнала $\widetilde{f_k}$ методом среднего гармонического взвешенного скользящего среднего:

$$\overline{f_k}(\alpha) = \left(\sum_{j=k-M}^{k+M} \frac{\alpha_{j+M+1-k}}{\widetilde{f_j}} \right)^{-1}$$

По варианту заданы метрики:

Евклидова для критерия зашумленности $\omega = \sqrt{\sum_{k=1}^{K} (\overline{f_k} - \overline{f_{k-1}})^2}$

Евклидова для критерия отличия $\delta = \sqrt{\frac{1}{K} \sum_{k=0}^{K} (\overline{f_k} - \widetilde{f_k})^2}$

Модель работы

$f_k = \sin(x_k) + 0.5$

$x_k = x_{min} + \frac{k(x_{max} - x_{min})}{K}$

$x_{min} = 0$, $x_{max} = \pi$

k = 0, … K, K = 100

амплитуда равномерного шума: 2a = 0.5;

дискретизация веса свертки: $\lambda_l = l/L$, l = 0, …, L, L = 10;

вероятность попадания в окрестность экстремума: P = 0.95;

интервал неопределенности: $\varepsilon = 0.01$;

размер скользящего окна: $r = 3$, $r^* = 5$;

Ход работы

Применяя случайный поиск для нахождения вектора α, прямой пассивный поиск – к поиску точки, максимально приближенной к точке утопии, можно усреднить значения, и тем самым приблизить зашумленный график к графику изначального сигнала.

Результаты работы программы:

Для окна скользящего окна размером r = 3:

```
Результат для для окна размером 3:

+----------+-------------------------------+
| Lambda   | 0.8                           |
| Alpha    | [0.292479 0.415043 0.292479] |
| W        | 0.690647                      |
| D        | 0.011879                      |
| J        | 0.554893                      |
| Distance | 0.690749                      |
+----------+-------------------------------+
```

Скриншот 1 - Результат для окна r = 3.

Значения функционала J и критериев ω, δ для оптимальных значений весов: J = 0.554893, $\omega = 0.690647$, $\delta = 0.011879$.

Оптимальное значение веса: $\lambda^* = 0.8$.

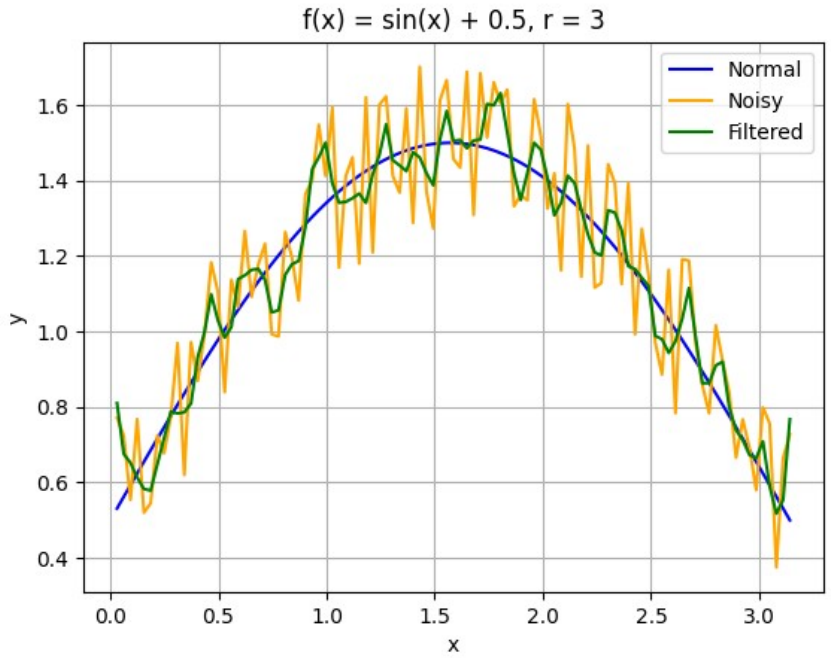

Рисунок 1 - Графики исходного сигнала, шума, очищенного сигнала для r = 3.

Таблица 1 - Результаты численного эксперимента для r = 3 и оптимальное значение веса λ*, функционала J и критериев ω, δ.

	Lambda	Alpha	W	D	J	Distance
0	0	[1.14000e-04 9.99773e-01 1.14000e-04]	2,29	0	0	2,29
1	0,1	[0.28832 0.423361 0.28832]	0,69	0,01	0,08	0,69
2	0,2	[0.289908 0.420184 0.289908]	0,69	0,01	0,15	0,69
3	0,3	[0.291335 0.41733 0.291335]	0,69	0,01	0,22	0,69
4	0,4	[0.292078 0.415845 0.292078]	0,69	0,01	0,28	0,69
5	0,5	[0.291943 0.416114 0.291943]	0,69	0,01	0,35	0,69
6	0,6	[0.29221 0.41558 0.29221]	0,69	0,01	0,42	0,69
7	0,7	[0.292221 0.415558 0.292221]	0,69	0,01	0,49	0,69
8	0,8	[0.292479 0.415043 0.292479]	0,69	0,01	0,55	0,69
9	0,9	[0.292321 0.415358 0.292321]	0,69	0,01	0,62	0,69
10	1	[0.292457 0.415086 0.292457]	0,69	0,01	0,69	0,69

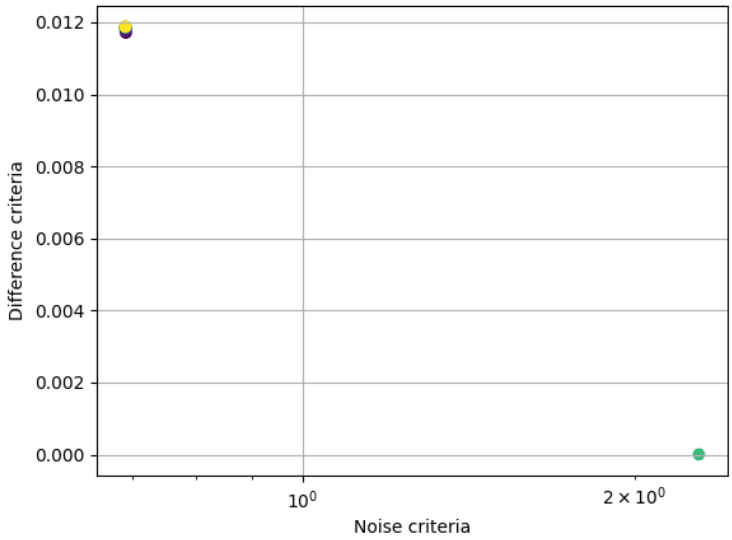

Рисунок 2 - Графическое отображение найденных приближений к оптимальным критериям в системе координат (ω, δ) в зависимости от весов λ для r = 3.

Для окна скользящего окна размером r = 5:

```
Результат для для окна размером 5:

+----------+-------------------------------------------------+
| Lambda   | 0.5                                             |
| Alpha    | [0.13824  0.22708  0.269359 0.22708  0.13824 ] |
| W        | 0.411007                                        |
| D        | 0.013034                                        |
| J        | 0.212021                                        |
| Distance | 0.411214                                        |
+----------+-------------------------------------------------+
```

Скриншот 2 - Результат для окна r = 3.

Значения функционала J и критериев ω, δ для оптимальных значений весов:

J = 0.212021, ω = 0.411007, δ = 0.013034.

Оптимальное значение веса: λ^* = 0.5.

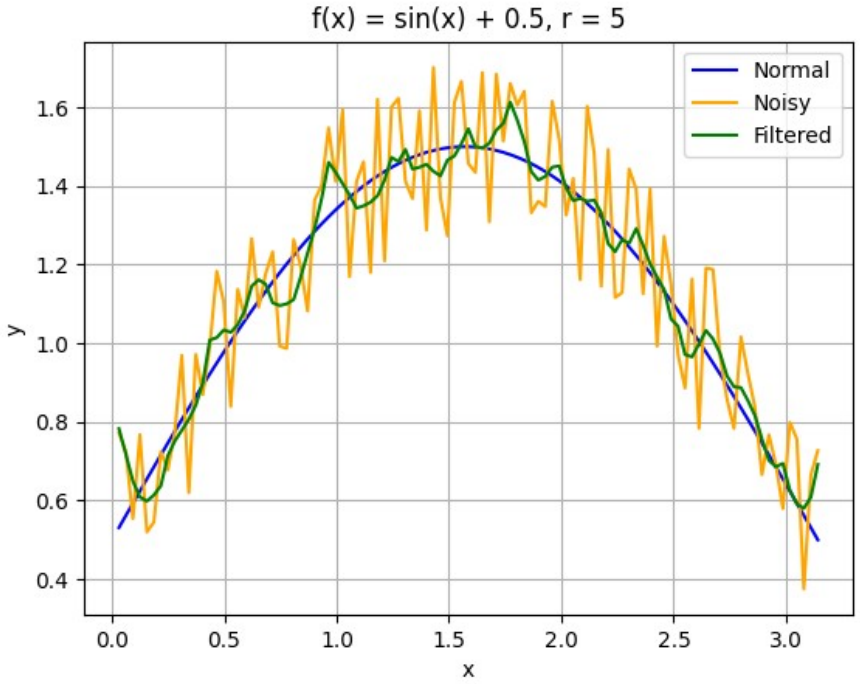

Рисунок 3 - Графики исходного сигнала, шума, очищенного сигнала для r = 5.

Таблица 2 - Графическое отображение найденных приближений к оптимальным критериям в системе координат (ω, δ) в зависимости от весов λ для r = 5.

	Lambda	Alpha	W	D	J	Distance
0	0	[3.80000e-05 7.60000e-05 9.99773e-01 7.60000e-05 3.80000e-05]	2,288615	0,000004	0,000004	2,288615
1	0,1	[0.137131 0.224557 0.276623 0.224557 0.137131]	0,411562	0,012903	0,052769	0,411764
2	0,2	[0.128244 0.225974 0.291565 0.225974 0.128244]	0,414252	0,012691	0,093003	0,414446
3	0,3	[0.142671 0.229978 0.254703 0.229978 0.142671]	0,413650	0,013281	0,133392	0,413863
4	0,4	[0.146297 0.222008 0.26339 0.222008 0.146297]	0,413281	0,013082	0,173161	0,413488
5	0,5	[0.13824 0.22708 0.269359 0.22708 0.13824]	0,411007	0,013034	0,212021	0,411214
6	0,6	[0.134063 0.233604 0.264666 0.233604 0.134063]	0,412356	0,013163	0,252678	0,412566
7	0,7	[0.125528 0.228627 0.29169 0.228627 0.125528]	0,414179	0,012713	0,293739	0,414374
8	0,8	[0.130472 0.229524 0.280007 0.229524 0.130472]	0,411444	0,012896	0,331735	0,411646
9	0,9	[0.128328 0.226637 0.290069 0.226637 0.128328]	0,413690	0,012719	0,373593	0,410000
10	1	[0.132846 0.226784 0.28074 0.226784 0.132846]	0,411554	0,012860	0,411554	0,411755

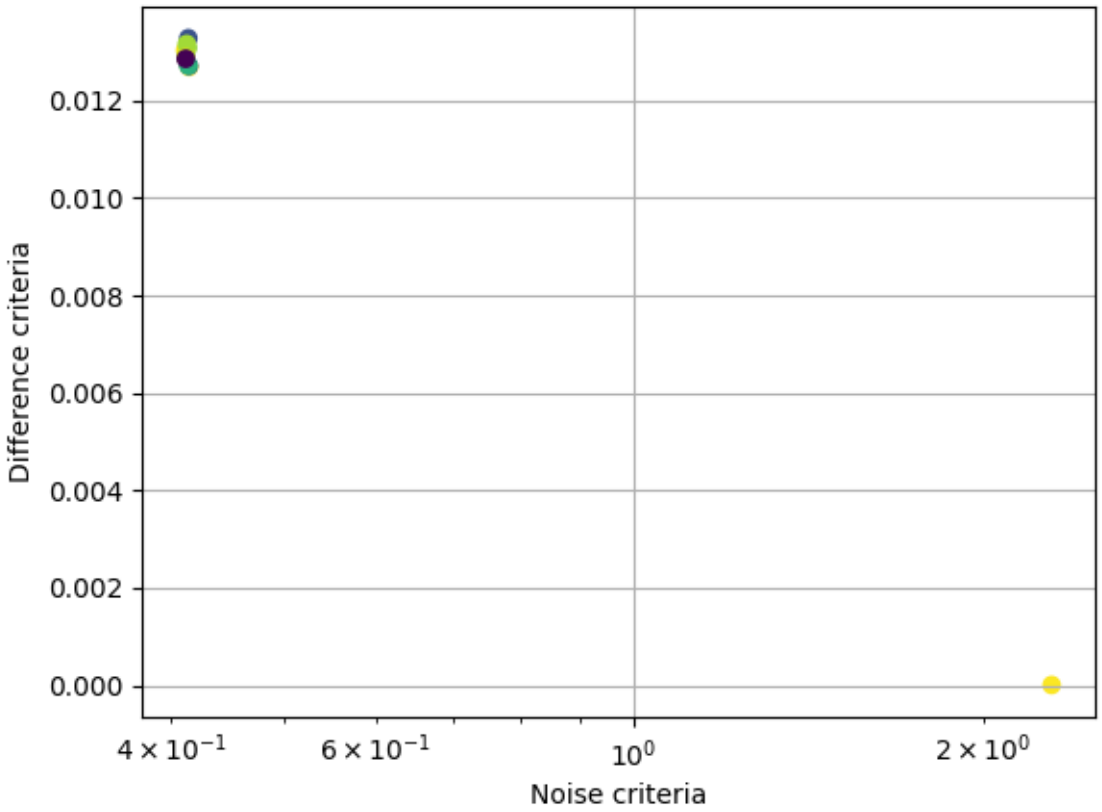

Рисунок 4 - Графическое отображение найденных приближений к оптимальным критериям в системе координат (ω, δ) в зависимости от весов λ для r = 5.

Выводы

Получив результаты работы алгоритма можно сделать вывод, что методы случайного и прямого пассивного поиска могут применяться в задачах многокритериальной оптимизации (фильтрации сигнала методом взвешенного скользящего среднего).

Приложение

Файл main.py:

```
import matplotlib.pyplot as plt
import pandas as pd
from tabulate import tabulate
import functions as fn
import numpy as np

"""
                    Лабораторная работа №7
        Исследование стохастической фильтрации сигналов как задачи
            двухкритериальной оптимизации с использованием методов прямого поиска

    Цель работы: Изучение основных принципов многокритериальной оптимизации в комбинации с методами случайного
        и прямого пассивного поиска применительно к задаче фильтрации дискретного сигнала методом
            взвешенного скользящего среднего.

                    Вариант 7
"""


# Функция формирования таблицы.
def create_table_final(_data):
    pd.set_option('display.width', None)
    table = pd.DataFrame(data=_data)
```

```
    print(tabulate(table, tablefmt='psql'), end='\n\n')


# Функция построения графиков функций.
def plot_results(x_source, y_source, y_noisy, y_filtered, title, filename):
    plt.title(title)
    plt.plot(x_source, y_source, color='blue')
    plt.plot(x_source, y_noisy, color='orange')
    plt.plot(x_source, y_filtered, color='green')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.grid()
    plt.legend(['Normal', 'Noisy', 'Filtered'])
    plt.savefig(filename)
    plt.clf()


# Графическое отображение найденных приближений
# к оптимальным критериям в системе координат (noise, difference)
def plot_criteria(noise, difference, filename):
    plt.xlabel('Noise criteria')
    plt.xscale('log')
    plt.ylabel('Difference criteria')
    colors = np.random.rand(len(noise))
    plt.scatter(noise, difference, c=colors)
    plt.grid()
    plt.savefig(filename)
    plt.clf()
```

```
if __name__ == '__main__':
    # Начальная функция и зашумленная функция.
    function = lambda x: np.sin(x) + 0.5
    noised_function = fn.uniform_noise(function, amplitude=0.5)

    # Первоначальные данные:
    # l = 0, ..., L, L = 10
    arrangement = (0., 1., 11)

    # Количество попыток.
    tries = fn.tries_count(x_min=0., x_max=np.pi, eps=0.01, probability=0.95)

    # Генерация интервала.
    x_source = fn.gen_x()

    # Значения функций.
    y_source = function(x_source)
    y_noised = noised_function(x_source)

    # Основная часть:
    # Для r = 3.
    index, results = fn.find_weights(y_noised, 3, tries, arrangement)
    results.to_csv('results_3.csv')
    weights = results.loc[index]['Alpha']
    y_filtered = fn.mean_harmonic_window(y_noised, weights)

    # Строим графики для окна r = 3.
    plot_results(x_source, y_source, y_noised, y_filtered, 'f(x) = sin(x) + 0.5, r = 3',
'results_3.png')
    plot_criteria(results['W'].values, results['D'].values, 'criteria_3.png')
```

```
    # Итоговый результат для окна размером 3.
    print('\nРезультат для для окна размером 3:', end='\n\n')

    create_table_final(results.loc[index])

    print('+=========+==========+==========+==========+==========+==========+==========+')
    print('+=========+==========+==========+==========+==========+==========+==========+', end='\n\n')

    # Для r = 5.
    index, results = fn.find_weights(y_noised, 5, tries, arrangement)
    results.to_csv('results_5.csv')
    weights = results.loc[index]['Alpha']
    y_filtered = fn.mean_harmonic_window(y_noised, weights)

    # Строим графики для окна r = 5.
    plot_results(x_source, y_source, y_noised, y_filtered, 'f(x) = sin(x) + 0.5, r = 5', 'results_5.png')
    plot_criteria(results['W'].values, results['D'].values, 'criteria_5.png')

    # Итоговый результат для окна размером 5.
    print('Результат для для окна размером 5:', end='\n\n')
    create_table_final(results.loc[index])
```

Файл functions.py:

```
import numpy as np
import pandas as pd
import random

SEED = 41
# SEED = 6446



# Количество попыток.
def tries_count(x_min, x_max, eps, probability):
    return int(np.log(1 - probability) / np.log(1 - eps / (x_max - x_min)))



# Вычисление евклидова расстояния.
def euclid_dist(f_array, s_array):
    result = np.subtract(f_array, s_array)
    return np.sqrt(np.sum(result ** 2))



# Критерий зашумленности.
def euclid_noise_criterion(array):
    return euclid_dist(array[1:], array[:-1])



# Критерий отличия.
def euclid_diff_criterion(f_array, s_array):
    assert len(f_array) == len(s_array)
    result = euclid_dist(f_array, s_array)
```

```
    return result / len(f_array)



# Генерация интервала.
def gen_x(x_max=0, x_min=np.pi, K=101):
    return [(x_min + k * (x_max - x_min) / K) for k in range(0, K)]



# Равномерное зашумливание.
def uniform_noise(func, amplitude, seed=SEED):
    assert amplitude > 0
    # Задаем амплитуду равномерного шума.
    # 2a = 0.5 -> a = 0.5 / 2
    amplitude /= 2
    rnd_gen = np.random.default_rng(seed=seed)
        return lambda x: np.add(func(x), rnd_gen.uniform(-amplitude, amplitude,
len(x)))



# Генератор окна скольжения.
def gen_window(window_size=3, seed=SEED):
    assert window_size > 1 and window_size % 2 == 1
    # Инициализация рандомайзера.
    random.seed(seed)
    while True:
        # Центральный вес.
        mid = random.uniform(0, 1)
        # Массив весов.
        w = []
```

```
        for _ in range(0, (window_size - 1) // 2 - 1):
            w.append(0.5 * random.uniform(0, 1 - mid - 2 * sum(w)))

        w.append(0.5 * (1 - mid - 2 * sum(w)))

        yield [*reversed(w), mid, *w]


# Скользящее среднее (среднее гармоническое).
def mean_harmonic_window(y_noisy, weights):
    m = (len(weights) - 1) // 2
    y_padded = np.pad(y_noisy, (m, m), 'constant', constant_values=1)
    return [np.sum([(w / y_padded[i + j]) for j, w in enumerate(weights)]) ** -1 for
i, _ in enumerate(y_noisy)]


# Функция случайного поиска.
def random_search(y_noisy, window, tries, lam):
    weights = [next(window) for _ in range(tries)]

    y_filtered = [mean_harmonic_window(y_noisy, w) for w in weights]

    noise_criterion = [euclid_noise_criterion(y_a) for y_a in y_filtered]

    diff_criterion = [euclid_diff_criterion(y_a, y_noisy) for y_a in y_filtered]

    # n_c - omega, d_c - delta.
    # omega - критерий зашумленности.
    # delta - критерий отличия.
```

```
    # lin_convolution - линейная свертка.
        lin_convolution = [lam * n_c + (1. - lam) * d_c for n_c, d_c in
zip(noise_criterion, diff_criterion)]

    index_min = np.argmin(lin_convolution, axis=0)

     return [np.round(weights[index_min], 6), np.round(noise_criterion[index_min],
6),
                                                np.round(diff_criterion[index_min],     6),
np.round(lin_convolution[index_min], 6)]



def find_weights(y_noisy, window_size, tries, arrangement, seed=SEED):
    result = []

    weights = gen_window(window_size, seed)
    for lam in np.linspace(*arrangement):
            result.append([np.round(lam, 1), *random_search(y_noisy, weights, tries,
lam)])
    result = pd.DataFrame(data=result, columns=['Lambda', 'Alpha', 'W', 'D', 'J'])

    # Считаем расстояние до идеальной точки.
    distance = np.sqrt(result['W'] ** 2 + result['D'] ** 2)

    result['Distance'] = np.round(distance, 6)
    return np.argmin(np.round(distance, 6)), result
```