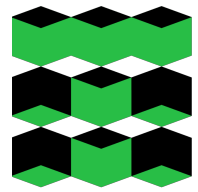
**Министерство науки и высшего образования Российской Федерации**

Федеральное государственное автономное образовательное учреждение

высшего образования



**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**

**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная школа информационных технологий и робототехники

Отделение информационных технологий

Направление подготовки 09.04.04 Программная инженерия

**Отчёт по лабораторной работе №1**

**МЕТОДЫ ФИЛЬТРАЦИИ СИГНАЛОВ**

по дисциплине Основы теории управления автономными системами

Выполнил студент гр. 8ПМ4Л \_\_\_\_\_\_\_ \_\_\_\_\_\_ Сокуров Р.Е.

Подпись Дата Фамилия И.О.

Проверил доцент ОАР \_\_\_\_\_\_\_ \_\_\_\_\_\_ Хожаев И.В.

Подпись Дата Фамилия И.О.

Томск 2024 г.

**Цель**

Изучить основные методы фильтрации измерительных сигналов.

**Задачи**

1) Изучить принцип работы следующих фильтров:

– фильтра экспоненциального сглаживания;

– фильтра скользящего среднего;

– медианного фильтра;

– фильтра с ограничением скорости нарастания сигнала;

2) Сгенерировать полезный сигнал и добавить к нему равномерно распределенный шум и всплески большой амплитуды;

3) Реализовать каждый из ранее перечисленных фильтров любым известным способом, проверить работоспособность фильтров на сгенерированном ранее зашумленном сигнале и изучить влияние настроечного параметра фильтра на качество обработки сигнала;

4) Подобрать комбинацию фильтров, отделяющих полезный сигнал от шума обоих типов; сравнить исходный полезный сигнал и отфильтрованный;

5) Оформить отчет.

**Ход работы**

**1. Генерация полезного сигнала и добавление к нему равномерно распределённого шума и всплесков большой амплитуды.**

В качестве генерируемого сигнала была выбрана функция . Данная функция была рассчитана на массиве , состоящем из 800 элементов. Затем, с помощью библиотеки «NumPy» языка «Python» был добавлен равномерно распределённый шум, а также высокоамплитудные случайные всплески (Рисунок 1). Код отображён в файле «code.py» в приложении А.

Изображение выглядит как линия, График, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 1 – Генерация зашумлённого сигнала

**2. Реализация фильтра экспоненциального сглаживания.**

Фильтр экспоненциального сглаживания – простейший фильтр низких частот, выходной сигнал которого равен взвешенной сумме последнего измеренного значения сигнала и значения отфильтрованного сигнала на предыдущем шаге:

,

где  – измеренные значения (зашумлённый сигнал),  – значения отфильтрованного сигнала,  – настроечный параметр фильтра.

Данный фильтр был реализован в функции «Exponential\_smoothing\_filter» в файле «code.py» приложения А.

Данная функция была использована с тремя разными значениями коэффициента :

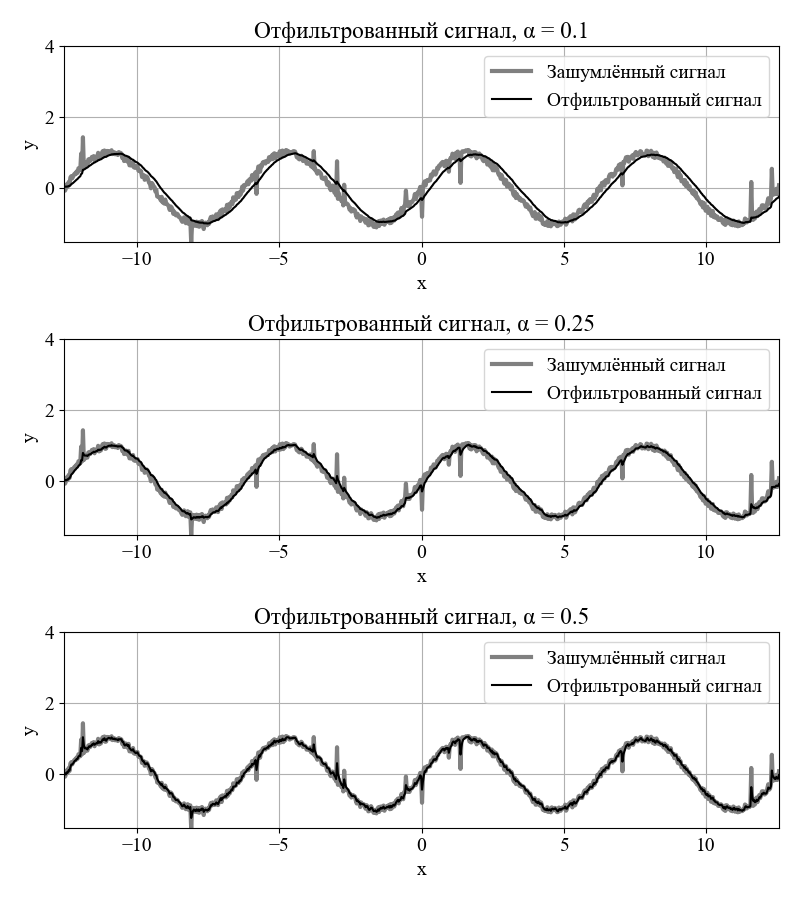


Рисунок 2 – Демонстрация работы фильтра экспоненциального сглаживания

По итогам анализа полученных графиков, оптимальным было принято значение , поскольку данное значение значительно сглаживает высокочастотные помехи в сигнале, но ещё не создаёт столь значимое запаздывание, как, например, значение .

**3. Реализация фильтра скользящего среднего**

Фильтр скользящего среднего является фильтром низких частот. Он хранит последние  измерений зашумленного сигнала и выдает на выход их среднее арифметическое.

Данный фильтр был реализован в функции «moving\_average\_filter» в файле «code.py» приложения А, значение :

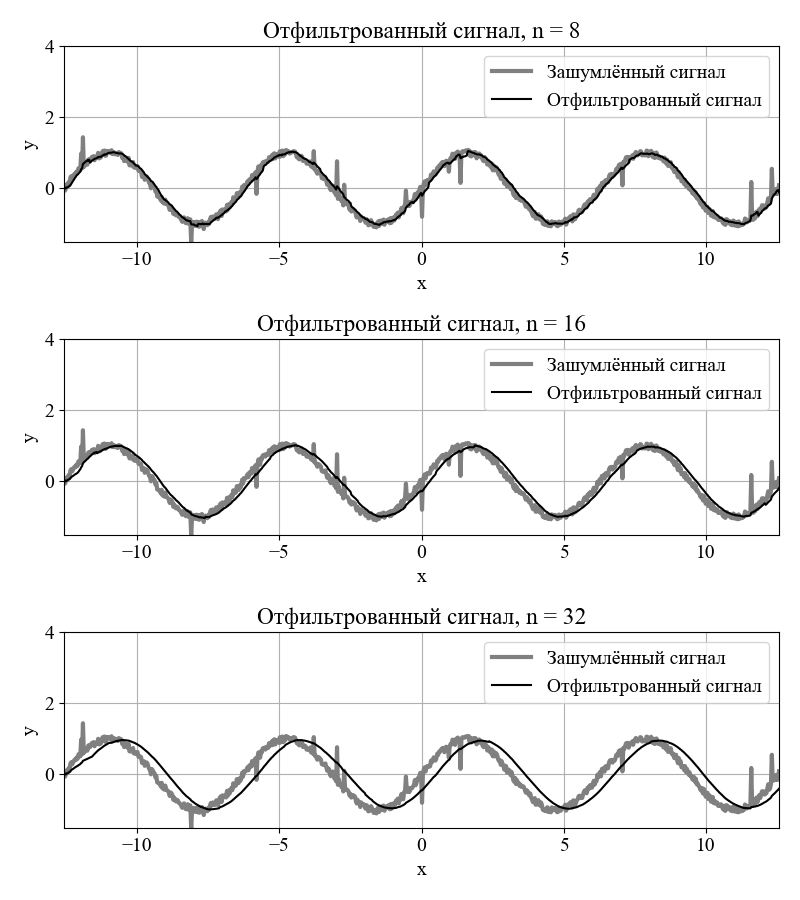


Рисунок 3 – Демонстрация работы фильтра скользящего среднего

Здесь наилучшим значением показал себя размер окна, равный 16, поскольку отставание отфильтрованного сигнала по вазе становится уже слишком большим при , а при  видны участки с резкими переходами.

**4. Реализация медианного фильтра**

Медианный фильтр является фильтром высоких частот. Его выходной сигнал равен медиане последних  измерений фильтруемого сигнала. Степень сглаживания сигнала также определяется шириной окна .

Данный фильтр был реализован в функции «median\_filter» в файле «code.py» приложения А, значение :

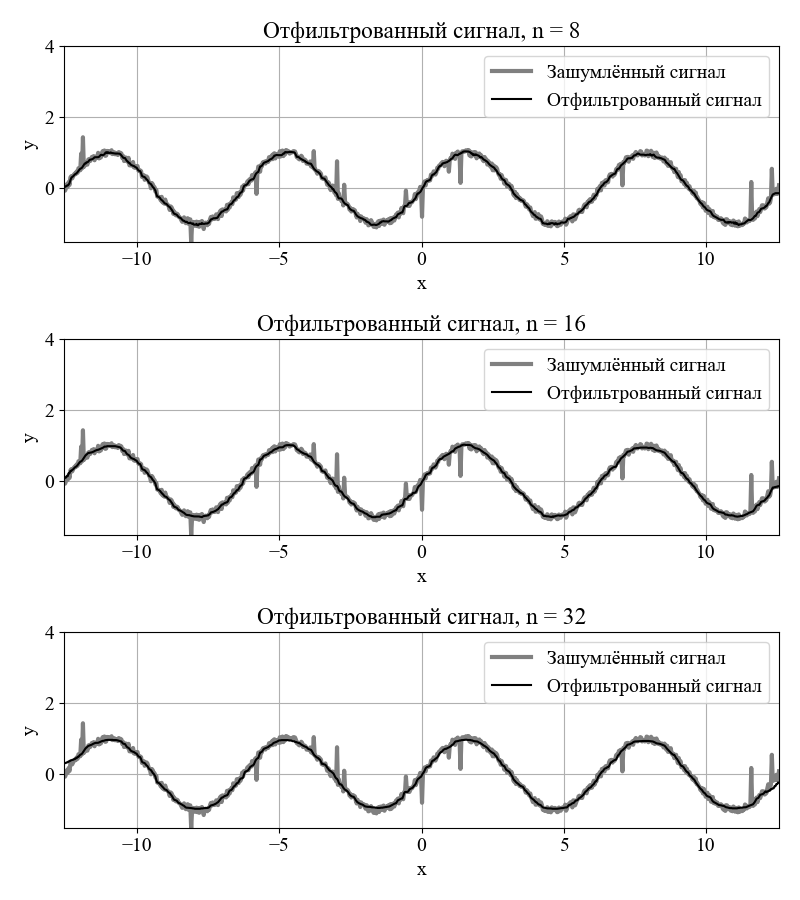


Рисунок 4 – Демонстрация работы медианного фильтра

Здесь, уже при размере окна  фильтр уже справился с низкими частотами (высокоамплитудных всплесков в итоговом сигнале нет), однако лучшим всё же показался размер окна , хотя стоит учесть, что фильтр может выдавать некорректные значения в начале (до того, как заполнится размер окна).

**5. Реализация фильтра с ограничением скорости нарастания сигнала**

Фильтр с ограничением скорости нарастания сигнала выводит на  
выход значение входного сигнала без изменений, если приращение между  
текущим и предыдущим его значением меньше заданного порогового  
значения. В противном случае значение выходного сигнала фильтра не  
изменяется.

Данный фильтр был реализован в функции «rate\_limit\_filter» в файле «code.py» приложения А, значение :

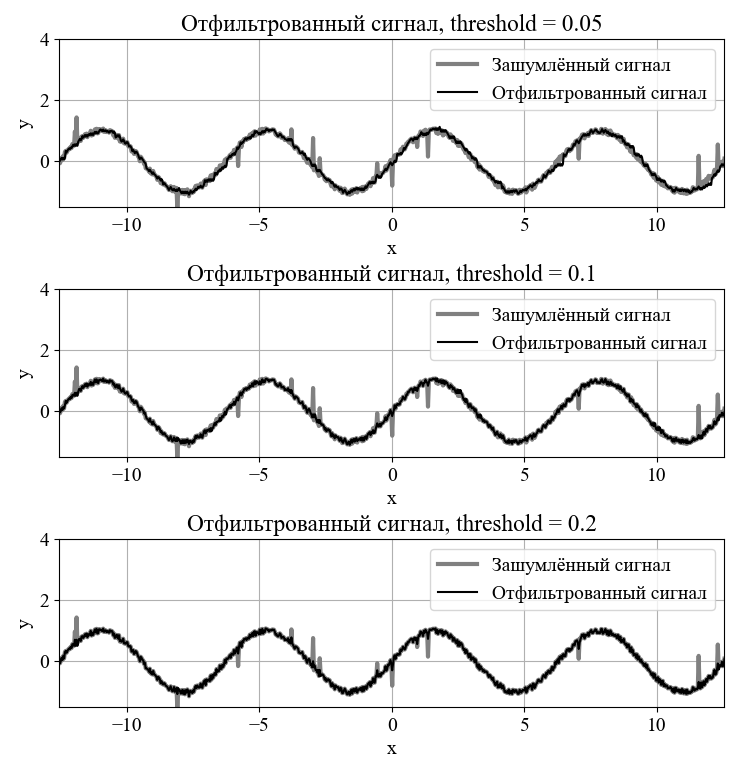


Рисунок 5 – Демонстрация работы фильтра с ограничением скорости нарастания

В данном фильтре, в отличие от всех предыдущих, именно увеличение значения  ведёт к большей схожести с оригинальным сигналом. Наилучшим значением было принято , поскольку при  уже появляются искажение сигнала.

**6. Реализация ансамбля (комбинации) фильтров**

Для реализации комбинации фильтров, которые лучше всех воспроизводят исходный сигнал из зашумленного, были выбраны медианный фильтр с  и фильтр экспоненциального сглаживания с :

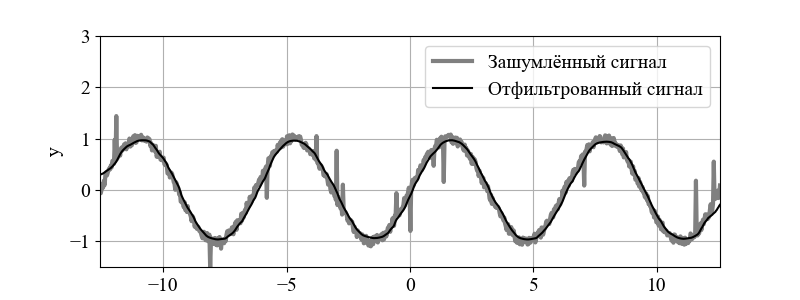


Рисунок 6 – Отфильтрованный двумя фильтрами сигнал

Несмотря на отклонение значений в начале и в конце (из-за особенностей работы медианного фильтра), данный ансамбль отлично справляется со своей задачей фильтрации, при условии, что интересующая нас информация находится не на краях графика.

# Заключение

В ходе данной лабораторной работы был составлен полезный сигнал , из которого, путём искусственного зашумления, получили материал для дальнейшей обработки фильтрами. Были исследованы 4 вида фильтров: фильтр экспоненциального сглаживания, фильтр скользящего среднего, медианный фильтр, фильтр с ограничением скорости нарастания сигнала. Каждый из них рассматривался при различных значениях настраиваемых параметров.

По итогам данных исследований было выбрано два фильтра, комбинация которых использовалась в п.6 работы для максимального приближения зашумлённого сигнала к исходному.

Приложение А

Реализация фильтров

1. # Подключение необходимых библиотек

2. import numpy as np

3. import matplotlib.pyplot as plt

4.

5. # Установление зерна случайно генерации

6. # Для возобнавляемости экспериментов

7. np.random.seed(52)

8.

9. # Постоянные переменыые

10. ALPHAS = [0.1, 0.25, 0.5] # Степень влияния прошлых значений для Exponential\_smoothing\_filter

11. N = [8, 16, 32] # Используется в скользящем среднем и медианном фильтрах

12. THRESHOLDS = [0.05, 0.1, 0.2] # Порог скорости нарастания сигнала

13.

14. # Настройки шрифта для графика

15. plt.rcParams['font.family'] = 'Times New Roman'

16. plt.rcParams['font.size'] = 14

17.

18. # ------------------------------------- Создание функции каждого фильтра -----------------------------------#

19. def Exponential\_smoothing\_filter(data, alpha):

20. """

21. Функция реализации фильтра экспоненциального сглаживания

22.

23. :param a: массив данных, которые нужно отфильтровать

24. :param alpha: регулировочный коэффициент

25. :return: отфильтрованный массив

26. """

27. filtered\_data = data.copy()

28. for i in range(1, filtered\_data.size):

29. filtered\_data[i] = alpha \* filtered\_data[i] + (1-alpha) \* filtered\_data[i-1]

30. return filtered\_data

31.

32. def moving\_average\_filter(data, n):

33. """

34. Функция реализации фильтра скользящего среднего

35.

36. :param data: массив данных, которые нужно отфильтровать

37. :param n: ширина окна усреднения

38. :return: отфильтрованный массив

39. """

40. # Массив для хранения отфильтрованных значений

41. filtered\_data = []

42.

43. # Массив для хранения текущего окна измерений

44. current\_values = []

45.

46. # Пройдем по всем элементам исходного массива

47. for new\_value in data:

48. # Добавляем текущее новое значение в массив измерений

49. current\_values.append(new\_value)

50.

51. # Если количество элементов меньше n, считаем среднее по всем доступным значениям

52. if len(current\_values) < n:

53. avg = np.mean(current\_values)

54. else:

55. # Как только элементов стало n или больше, считаем среднее по последним n значениям

56. avg = np.mean(current\_values[-n:])

57.

58. # Добавляем отфильтрованное значение в результат

59. filtered\_data.append(avg)

60.

61. return filtered\_data

62.

63. def median\_filter(data, window\_size):

64. """

65. Функция реализации медианного фильтра

66.

67. :param data: массив данных, которые нужно отфильтровать

68. :param n: ширина окна расчёта медианы

69. :return: отфильтрованный массив

70. """

71. # Если размер окна меньше 1 или больше длины сигнала, возвращаем исходный сигнал

72. if window\_size < 1 or window\_size > len(data):

73. return data

74.

75. # Создаем список для хранения отфильтрованных значений

76. filtered\_data = []

77.

78. half\_window = window\_size // 2

79.

80. # Проходим по каждому элементу сигнала

81. for i in range(len(data)):

82. # Выбираем окно вокруг текущего элемента

83. start\_idx = max(0, i - half\_window)

84. end\_idx = min(len(data), i + half\_window + 1)

85.

86. # Получаем подмассив, сортируем его и находим медиану

87. window = sorted(data[start\_idx:end\_idx])

88.

89. # Если окно нечетное, выбираем центральный элемент

90. if window\_size % 2 == 1:

91. median = window[len(window) // 2]

92. else:

93. # Если окно четное, берем среднее арифметическое двух центральных элементов

94. mid1 = window[len(window) // 2 - 1]

95. mid2 = window[len(window) // 2]

96. median = (mid1 + mid2) / 2.0

97.

98. # Добавляем медиану в отфильтрованный сигнал

99. filtered\_data.append(median)

100.

101. return np.array(filtered\_data)

102.

103. def rate\_limit\_filter(data, threshold):

104. """

105. Функция реализации функции ограниченной скорости нарастания сигнала

106.

107. :param data: массив данных, которые нужно отфильтровать

108. :param threshold: пороговое значение скорости нарастания

109. :return: отфильтрованный массив

110. """

111.

112. # Инициализация массива выходного сигнала

113. filtered\_data = [data[0]]

114.

115. # Применение фильтра

116. for i in range(1, len(data)):

117. delta = data[i] - data[i-1]

118. if abs(delta) <= threshold:

119. filtered\_data.append(data[i])

120. else:

121. # Ограничение прироста

122. if delta > 0:

123. filtered\_data.append(filtered\_data[-1] + threshold)

124. else:

125. filtered\_data.append(filtered\_data[-1] - threshold)

126.

127. return filtered\_data

128.

129. # --------------------- Создание чистого и зашумленного сигналов для дальнейших эксериментов -------------------#

130. x = np.linspace(-4\*np.pi, 4\*np.pi, 800) # Создание массива иксов

131. y = np.sin(x) # Функция

132. y\_clean = np.sin(x)

133. noise = np.random.uniform(-0.1,0.1,800) # Добавляем равномерно распределенный шум

134. spikes = np.random.choice([0, 1], size=x.shape, p=[0.98, 0.02]) # Случайные всплески (2% вероятность)

135. amplitude = np.random.uniform(-1, 1, size=x.shape) # Случайная амплитуда всплесков

136. y += noise + spikes \* amplitude # дальше наш сигнал это зашумлённый сигнал

137.

138. # --------------------- Вывод на график результата фильтрации (нужный раскоментить) ----------------------------#

139.

140. #y\_exp = Exponential\_smoothing\_filter(y)

141. # plt.figure(figsize=[8,3])

142. # plt.plot(x,y, color='gray', linewidth=3, label='Зашумлённый сигнал')

143. # plt.plot(x,y\_clean, color='black', linewidth=1.5, label='Оригинальный сигнал')

144. # plt.grid()

145. # ax = plt.gca()

146. # ax.set\_xlim([-4\*np.pi,4\*np.pi])

147. # ax.set\_ylim([-1.5,3])

148. # plt.xlabel('x')

149. # plt.ylabel('y')

150. # plt.legend()

151.

152. # --------------------- Вывод на график результата фильтрации (нужный раскоментить) ----------------------------#

153.

154. # plt.figure(figsize=[8, 9]) # Размер всей фигуры: 8x9 (3 графика по 8x3)

155.

156. # for idx, alpha in enumerate(ALPHAS):

157. # y\_exp = Exponential\_smoothing\_filter(y, alpha)

158. # plt.subplot(len(ALPHAS), 1, idx + 1)

159. # plt.plot(x, y, color='gray', linewidth=3, label='Зашумлённый сигнал')

160. # plt.plot(x, y\_exp, color='black', linewidth=1.5, label=f'Отфильтрованный сигнал')

161. # plt.grid()

162. # plt.xlim([-4 \* np.pi, 4 \* np.pi])

163. # plt.ylim([-1.5, 4])

164. # plt.xlabel('x')

165. # plt.ylabel('y')

166. # plt.legend()

167. # plt.title(f'Отфильтрованный сигнал, α = {alpha}')

168.

169. # plt.tight\_layout()

170.

171. # --------------------- Вывод на график результата фильтрации (нужный раскоментить) ----------------------------#

172.

173. # plt.figure(figsize=[8, 9]) # Размер всей фигуры: 8x9 (3 графика по 8x3)

174.

175. # for idx, n in enumerate(N):

176. # y\_exp = moving\_average\_filter(y, n)

177. # plt.subplot(len(N), 1, idx + 1)

178. # plt.plot(x, y, color='gray', linewidth=3, label='Зашумлённый сигнал')

179. # plt.plot(x, y\_exp, color='black', linewidth=1.5, label=f'Отфильтрованный сигнал')

180. # plt.grid()

181. # plt.xlim([-4 \* np.pi, 4 \* np.pi])

182. # plt.ylim([-1.5, 4])

183. # plt.xlabel('x')

184. # plt.ylabel('y')

185. # plt.legend()

186. # plt.title(f'Отфильтрованный сигнал, n = {n}')

187.

188. # plt.tight\_layout()

189.

190. # --------------------- Вывод на график результата фильтрации (нужный раскоментить) ----------------------------#

191.

192. # plt.figure(figsize=[8, 9]) # Размер всей фигуры: 8x9 (3 графика по 8x3)

193.

194. # for idx, n in enumerate(N):

195. # y\_exp = median\_filter(y, n)

196. # plt.subplot(len(N), 1, idx + 1)

197. # plt.plot(x, y, color='gray', linewidth=3, label='Зашумлённый сигнал')

198. # plt.plot(x, y\_exp, color='black', linewidth=1.5, label=f'Отфильтрованный сигнал')

199. # plt.grid()

200. # plt.xlim([-4 \* np.pi, 4 \* np.pi])

201. # plt.ylim([-1.5, 4])

202. # plt.xlabel('x')

203. # plt.ylabel('y')

204. # plt.legend()

205. # plt.title(f'Отфильтрованный сигнал, n = {n}')

206.

207. # plt.tight\_layout()

208.

209. # --------------------- Вывод на график результата фильтрации (нужный раскоментить) ----------------------------#

210.

211. # plt.figure(figsize=[8, 9]) # Размер всей фигуры: 8x9 (3 графика по 8x3)

212.

213. # for idx, threshold in enumerate(THRESHOLDS):

214. # y\_exp = rate\_limit\_filter(y, threshold)

215. # plt.subplot(len(THRESHOLDS), 1, idx + 1)

216. # plt.plot(x, y, color='gray', linewidth=3, label='Зашумлённый сигнал')

217. # plt.plot(x, y\_exp, color='black', linewidth=1.5, label=f'Отфильтрованный сигнал')

218. # plt.grid()

219. # plt.xlim([-4 \* np.pi, 4 \* np.pi])

220. # plt.ylim([-1.5, 4])

221. # plt.xlabel('x')

222. # plt.ylabel('y')

223. # plt.legend()

224. # plt.title(f'Отфильтрованный сигнал, threshold = {threshold}')

225.

226. # plt.tight\_layout()

227.

228. # --------------------- Вывод на график результата фильтрации (нужный раскоментить) ----------------------------#

229.

230. y\_med = median\_filter(y, 32)

231. y\_exp = Exponential\_smoothing\_filter(y\_med, 0.25)

232. plt.figure(figsize=[8,3])

233. plt.plot(x,y, color='gray', linewidth=3, label='Зашумлённый сигнал')

234. plt.plot(x,y\_exp, color='black', linewidth=1.5, label='Отфильтрованный сигнал')

235. plt.grid()

236. ax = plt.gca()

237. ax.set\_xlim([-4\*np.pi,4\*np.pi])

238. ax.set\_ylim([-1.5,3])

239. plt.xlabel('x')

240. plt.ylabel('y')

241. plt.legend()

242.

243. plt.show()

244.