

Министерство науки и высшего образования Российской Федерации  
Новосибирский государственный технический университет  
Кафедра теоретической и прикладной информатики

## Планирование и анализ эксперимента

Курсовая работа по теме:

«Синтез дискретных  $\Lambda$ -оптимальных планов эксперимента  
для нечетких линейных двухфакторных моделей  
с сигмоидными функциями принадлежности»

Факультет:	ФПМИ
Группа:	ПМ-63
Студент:	Утюганов Д.С.
Вариант:	22

Новосибирск

2020

# 1. Задание

1. Ознакомиться с математическим аппаратом построения регрессионных моделей в рамках концепции нечетких систем, вопросами оптимального планирования эксперимента.

2. Разработать программное приложение синтеза дискретных А-оптимальных планов эксперимента для двухфакторных моделей. В качестве алгоритма использовать градиентный алгоритм **замены точек**.

3. Работа приложения должна быть продемонстрирована на нескольких тестовых примерах.

# 2. Требования

Приложение должно осуществлять синтез дискретных А-оптимальных планов эксперимента для нечетких регрессионных моделей с двумя вещественными факторами.

Область определения каждой вещественной переменной (интервал  $[-1; +1]$ ) при фаззификации разбивается на две сигмоидные нечеткие партии с функциями принадлежности:

$$\mu_2(x) = \frac{1}{1 + e^{-d_2(x-d_1)}}$$

$$\mu_1(x) = 1 - \mu_2(x)$$

Построить оптимальные планы для значений  $d_1 = 0; d_2 = \{8; 12; 16; 20\}$ . Базовая модель линейная. Область планирования - регулярная сетка  $21 \times 21$ . Число точек в плане 20, 30, 40. Характеристики построенных планов представить в таблице, координаты точек спектра планов отобразить на рисунке и в виде таблицы.

# 3. Анализ

## 3.1. Нечёткая логика

В исходном виде линейная двухфакторная модель по своему списку регрессоров имеет вид:

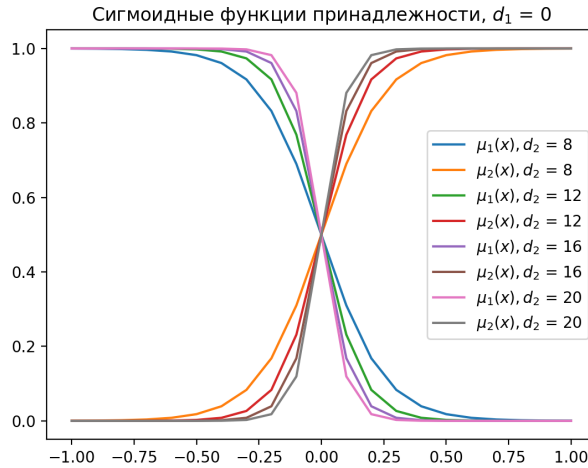
$$f^T(x) = (1, x_1, x_2, \mu_1(x_1), \mu_2(x_1), \mu_1(x_2), \mu_2(x_2),$$

$$\mu_1(x_1)x_1, \mu_2(x_1)x_1, \mu_1(x_2)x_1, \mu_2(x_2)x_1,$$

$$\mu_1(x_1)x_2, \mu_2(x_1)x_2, \mu_1(x_2)x_2, \mu_2(x_2)x_2)$$

Редуцируем регрессоры, связанные со **второй** партицией. Тогда модель имеет вид:

$$f^T(x) = (1, x_1, x_2, \mu_1(x_1), \mu_1(x_2), \mu_1(x_1)x_1, \mu_1(x_2)x_1, \mu_1(x_1)x_2, \mu_1(x_2)x_2)$$



### 3.2. Критерий А-оптимальности и проиводная

$$\varepsilon^* = \underset{\varepsilon}{\operatorname{Arg\,min}} \operatorname{tr} \left( M^{-1}(\varepsilon) \right)$$

$$\frac{\partial \Psi [M(\varepsilon)_N^s]}{\partial M(\varepsilon_N)} = \left( M^{-2}(\varepsilon) \right)^T$$

### 3.3. Градиентный алгоритм

1. Выбирается невырожденный начальный план  $\varepsilon_N^0$ ,  $s = 0$ .

2. Вычисляются элементы вектора градиента

$$\phi(x_j, \varepsilon_N^s) = f^T(x_j) \frac{\partial \Psi [M(\varepsilon)_N^s]}{\partial M(\varepsilon_N)} f(x_j), \quad j = 1 \dots n$$

для плана  $\varepsilon_N^s$ ,  $\varepsilon_{N,i}^s = \varepsilon_N^s$ . счётчик числа проведённых замен точек при движении по вычисленному направлению градиента устанавливается в 0 ( $i=1$ ).

3. Выбирается точка  $x^*$  на множестве  $\tilde{X}$  по правилу

$$x^* = \operatorname{Arg\,max}_{x \in \tilde{X}} \phi(x, \varepsilon_N^s)$$

4. Среди точек плана  $\varepsilon_{N,i}^s$  выбирается точка  $x^{**}$  по правилу

$$x^{**} = \operatorname{Arg\,min}_{x_j \in \varepsilon_{N,i}^s} \phi(x_j, \varepsilon_N^s)$$

5. Точка  $x^{**}$  заменяется в плане  $\varepsilon_{N,i}^s$  на точку  $x^*$ . В результате формируется план  $\varepsilon_{N,i+1}^s$ .

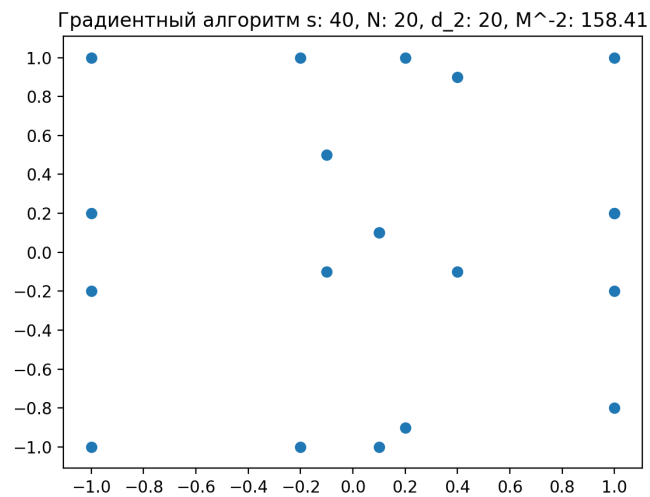
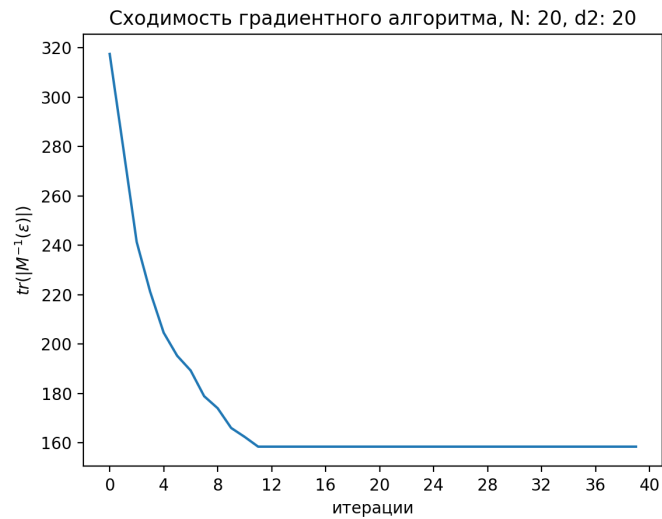
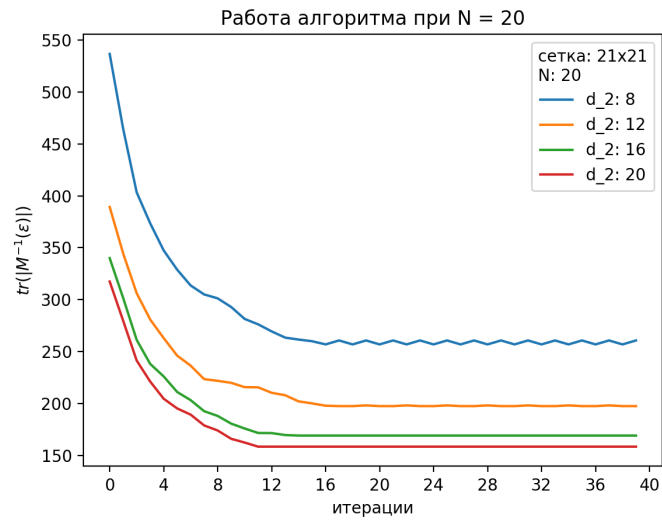
6. Сравниваются величины  $\Psi [M(\varepsilon_{N,i+1}^s)]$  и  $\Psi [M(\varepsilon_{N,i}^s)]$

а) если  $\Psi [M(\varepsilon_{N,i+1}^s)] > \Psi [M(\varepsilon_{N,i}^s)]$ , то счётчик  $i$  проведённых удачных замен точек увеличивается на единицу и осуществляется переход на шаг 3, при этом точки  $x^{**}$  и  $x^*$  исключаются из рассмотрения;

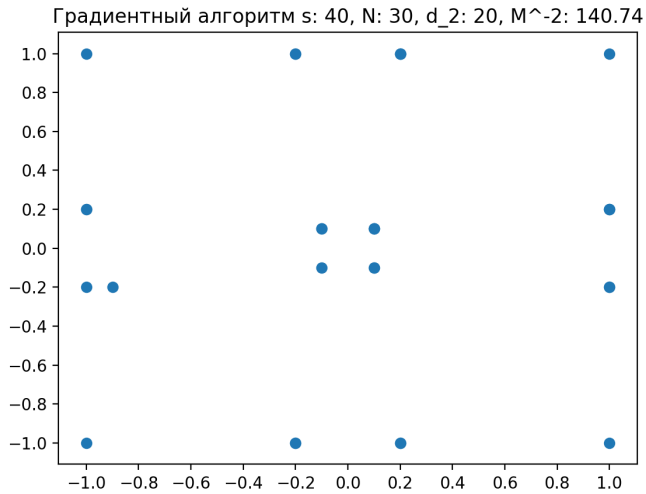
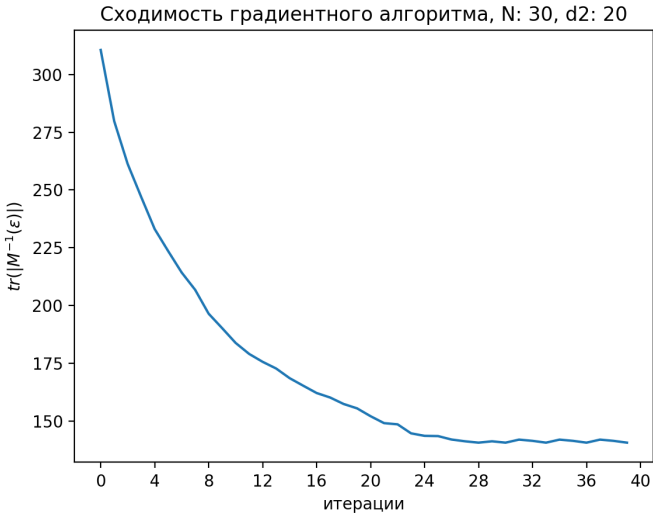
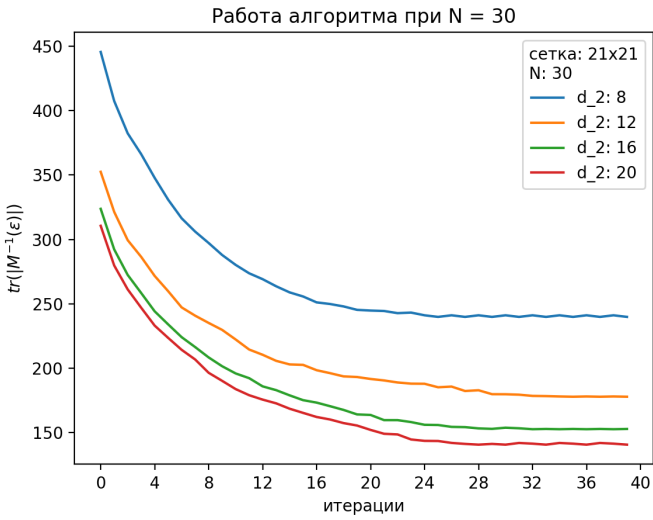
б) в противном случае: если  $i = 0$ , то вычисления прекращаются, иначе –  $s$  заменяется на  $s+1$  и осуществляется переход на шаг 2.

## 4. Исследование влияния числа точек плана N

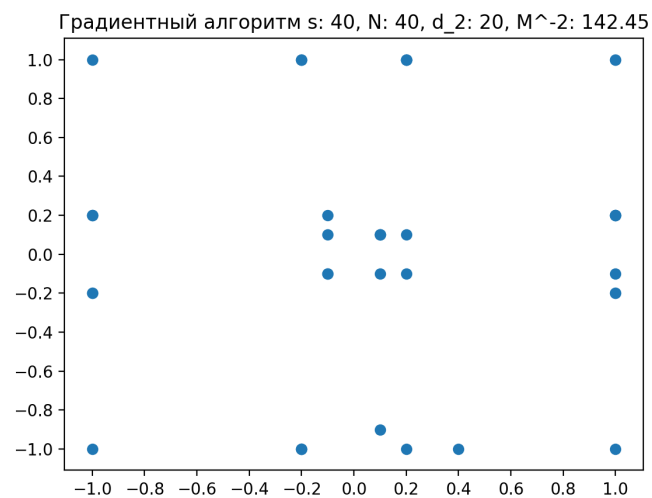
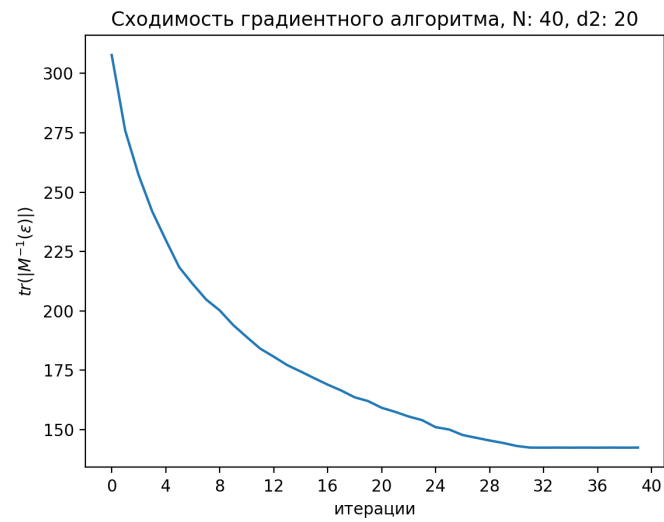
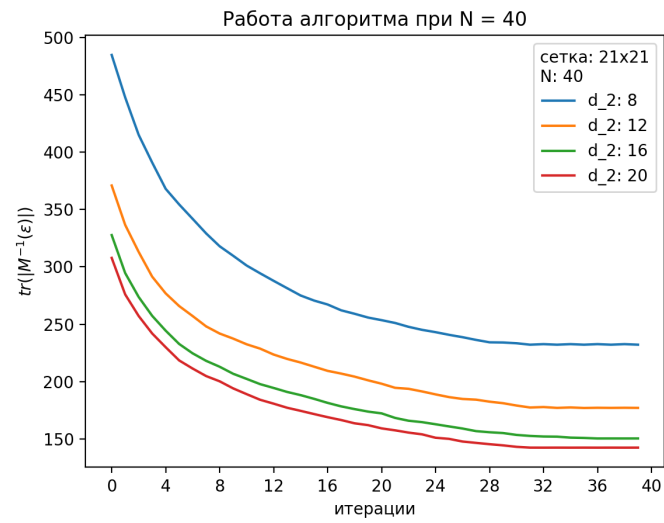
### 4.1. $N = 20$



4.2.  $N = 30$



### 4.3. $N = 40$

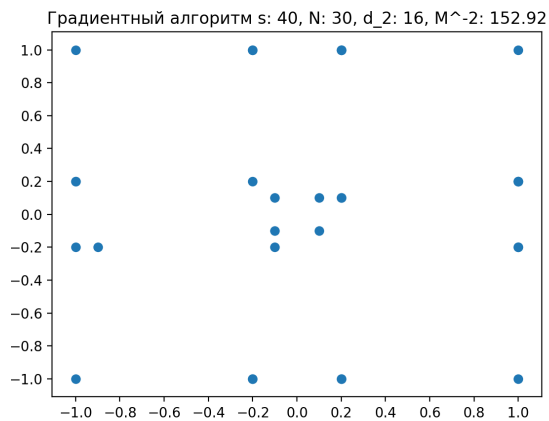
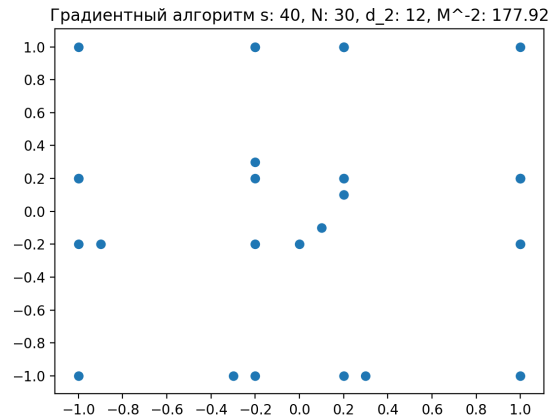
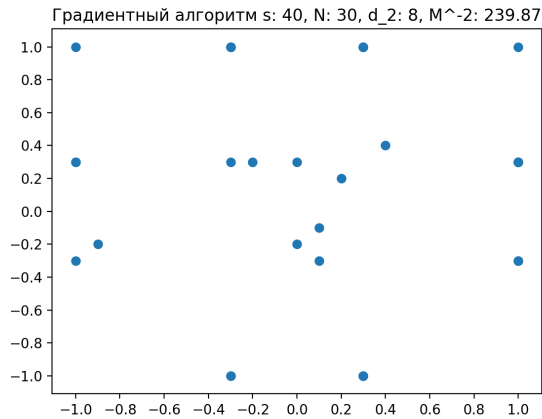


## 4.4. Вывод

С ростом числа точек плана от 20 до 40 план становится более А-оптимальным. На 30 итерации  $tr(M^{-1})$  значение падает со 160 до 140.

Можно заметить, что остались точки, с координатами близкими к -1, 0, 1.

## 5. Исследование параметра $d_2$



### 5.1. Вывод:

Как видно, при росте параметра  $d_2$  точки стремятся к осям и началу координат.

Критерий А-оптимальности плана уменьшилась почти в 2 раза с 239.87 до 140.74.

## 6. Исходный код программы

main.py

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from numpy.linalg import det, inv, norm
4
5
6 k = 2          # число переменных
7 m = 9          # число параметров
8 d_1 = 0.0
9 d_2 = 20.0
10 MAX_ITER = 40
11 t = np.linspace(0, MAX_ITER, 11)
12
13
14 # сигмоидная функция принадлежности
15 def mu_1(x):
16     return 1.0 - (1.0 / (1.0 + np.exp(-d_2*(x-d_1))))
17
18 def f_vector(x):
19     x1 = x[0]
20     x2 = x[1]
21     return np.array([
22         1],
23         [x1],
24         [x2],
25         [mu_1(x1)],
26         [mu_1(x2)],
27         [mu_1(x1)*x1],
28         [mu_1(x2)*x1],
29         [mu_1(x1)*x2],
30         [mu_1(x2)*x2]
31     ])
32
33 def f_vector_T(x):
34     x1 = x[0]
35     x2 = x[1]
36     return np.array([
37         1,
38         x1,
39         x2,
40         mu_1(x1),
41         mu_1(x2),
42         mu_1(x1)*x1,
43         mu_1(x2)*x1,
44         mu_1(x1)*x2,
45         mu_1(x2)*x2
46     ])
47
48
49 #
50 #
51 class Coursework():
52     def __init__(self, N):
53         ''' Выделение памяти под массивы '''
54         width = 21
55         n = width**2
56         self.x_grid = np.ndarray((n, k))
```



```

57     self.x_plan = np.ndarray((N, k))
58     self.M = np.ndarray((m, m))
59     self.D = np.ndarray((m, m))
60     self.width = width
61     self.n = n
62     self.N = N
63
64     def draw_plan_on_s(self, s, A):
65         alg_name = 'Градиентный алгоритм s: {}, N: {}, d_2: {:.0f}, M^-2: {:.2f}'.
66         format(s, self.N, d_2, A)
67         path = 'pics/plan_grad_alg_N_{s}_{d2}_{:.0f}.png'.format(self.N, s,
68         d_2)
69         t = np.linspace(-1, 1, 11)
70         plt.title(alg_name)
71         plt.scatter([self.x_plan[i][0] for i in range(len(self.x_plan))], [self.
72         x_plan[i][1] for i in range(len(self.x_plan))], )
73         plt.xticks(t)
74         plt.yticks(t)
75         plt.savefig(path, dpi=200)
76         plt.clf()
77
78     def generate_initial_guess(self):
79         ''' Задаём начальное приближение '''
80         # создаём сетку
81         t = np.linspace(-1, 1, self.width)
82         i = 0
83         for x1 in t:
84             for x2 in t:
85                 self.x_grid[i] = np.array([x1, x2])
86                 i+=1
87
88         # случайно выбираем точки плана и сохраняем
89         # for i in range(self.N):
90         #     self.x_plan[i] = self.x_grid[np.random.choice(self.n)]
91         # np.savetxt('plans/plan_{x}_{y}.txt'.format(self.width, self.width,
92         self.N), self.x_plan)
93
94         # или же загружаем
95         self.x_plan = np.loadtxt('plans/plan_{x}_{y}.txt'.format(self.width,
96         self.width, self.N), dtype=np.float)
97
98     def build_matrix_M(self):
99         ''' Построение информационной матрицы M без использования весов плана'''
100         self.M = np.zeros((m, m))
101         for i in range(self.N):
102             x = self.x_plan[i]
103             self.M += f_vector(x) * f_vector_T(x)
104         self.M /= self.N
105
106     def build_matrix_D(self):
107         ''' Построение дисперсионной матрицы D '''
108         self.D = inv(self.M)
109
110     def calc_A(self):
111         '''
112         Критерий A — оптимальности. (A — average variance)
113         Эллипсоид рассеивания с наименьшей суммой квадратов длин осей
114         '''
115         return np.trace(self.D)

```

```

112 def dPsi(self):
113     ''' d Psi(M) / d M '''
114     return np.transpose(self.D @ self.D)
115
116 #
117 #
118 def gradient_algorithm(self, do_visualisation = False):
119     '''
120     Градиентный алгоритмы синтеза дискретного
121     оптимального плана эксперимента
122     '''
123     # шаг 1
124     # задаём начальное приближение
125     self.generate_initial_guess()
126     do_calc = True
127     s = 0
128     A_prev = 0.0
129     A = 0.0
130     result = np.ndarray(MAX_ITER)
131
132     while do_calc == True and s < MAX_ITER:
133         # шаг 2
134         # вычисляем элементы вектора градиента
135         self.build_matrix_M()
136         self.build_matrix_D()
137
138         grad_plan = np.ndarray(self.N)
139         for i, x in enumerate(self.x_plan):
140             grad_plan[i] = f_vector_T(x) @ self.dPsi() @ f_vector(x)
141
142         grad_grid = np.ndarray(self.n)
143         for i, x in enumerate(self.x_grid):
144             grad_grid[i] = f_vector_T(x) @ self.dPsi() @ f_vector(x)
145
146         i_counter = 0 # счётчик успешных замен
147         do_replaces = True
148         N_indicies = {i for i in range(self.N)}
149         n_indicies = {i for i in range(self.n)}
150
151
152         while do_replaces and i_counter < self.N:
153             grad_grid = np.ndarray(self.n)
154             for i, x in enumerate(self.x_grid):
155                 grad_grid[i] = f_vector_T(x) @ self.dPsi() @ f_vector(x)
156
157             # шаг 3
158             # поиск x*
159             max_j = 0
160             max_val = grad_grid[max_j]
161             for j in n_indicies:
162                 if grad_grid[j] > max_val:
163                     max_val = grad_grid[j]
164                     max_j = j
165
166             # шаг 4
167             # поиск x**
168             min_i = 0
169             min_val = grad_plan[min_i]
170             for i in N_indicies:
171                 if grad_plan[i] < min_val:

```

```

172         min_val = grad_plan[i]
173         min_i = i
174
175     # шаг 5
176     # замена x** на x*
177     self.x_plan[min_i] = self.x_grid[max_j]
178
179     self.build_matrix_M()
180     self.build_matrix_D()
181     A_prev = A
182     A = self.calc_A()
183
184     # шаг 6
185     # сравниваем функционал на x2- итерациях
186     if A > A_prev:
187         i_counter += 1
188         N_indicies.remove(min_i)
189         n_indicies.remove(max_j)
190     else:
191         do_replaces = False
192     result[s] = A
193
194     if s % 10 == 0 and do_visualisation:
195         self.draw_plan_on_s(s, A)
196     s += 1
197     print('s: {}, Psi: {:.2f}'.format(s, A))
198
199     if do_calc == False:
200         for i in range(s, MAX_ITER):
201             result[i] = result[s-1]
202
203     if do_visualisation:
204         self.draw_plan_on_s(s, A)
205     return result
206
207
208 #
209 #
210 def perform_experiment(N, do_visualisation = False):
211     ''' Отдельный эксперимент '''
212     cw = Coursework(N)
213     return cw.gradient_algorithm(do_visualisation)
214
215 def research_N():
216     ''' Исследование работы алгоритма при различных N '''
217     print('Исследование работы алгоритма при различных N')
218     d1 = 0
219     for N in [20, 30, 40]:
220         for d2 in [8, 12, 16, 20]:
221             global d_1, d_2
222             d_1 = float(d1)
223             d_2 = float(d2)
224             y = perform_experiment(N, False)
225             plt.plot(y, label='d_2: {:.0f}'.format(d2))
226             plt.title('Работа алгоритма при N = {}'.format(N))
227             plt.legend(title='сетка: 21x21\nN: {}'.format(N))
228             plt.xticks(t)
229             plt.xlabel('итерации')
230             plt.ylabel(r'$\text{tr}(\left| M^{-1}(\varepsilon) \right|)$')
231             plt.savefig('pics/research_N_{}.png'.format(N), dpi=200)

```

```

232     plt.clf()
233
234 def research_d2():
235     ''' Исследование работы алгоритма при различных d2 '''
236     print('Исследование работы алгоритма при различных d2')
237     N = 30
238     d1 = 0
239     for d2 in [8, 12, 16, 20]:
240         global d_1, d_2
241         d_1 = float(d1)
242         d_2 = float(d2)
243         y = perform_experiment(N, True)
244         plt.plot(y, label='d_2: {:.0f}'.format(d2))
245     plt.title(r'Работа алгоритма при N = {}'.format(N))
246     plt.legend(title='сетка: 21x21\nN: {}'.format(N))
247     plt.xticks(t)
248     plt.xlabel('итерации')
249     plt.ylabel(r'$\left| M^{-1}(\varepsilon) \right|$')
250     plt.savefig('pics/research_d2_N_{}.png'.format(N), dpi=200)
251     plt.clf()
252
253 def show_convergence_of_grad_alg(N, do_visualisation = False):
254     ''' Отрисовка сходимости градиентного алгоритма '''
255     print('Отрисовка сходимости градиентного алгоритма')
256     title = 'Сходимость градиентного алгоритма, N: {}, d2: {:.0f}'.format(N, d_2)
257     path = 'pics/convergence_grad_alg_N_{}_d2_{:.0f}.png'.format(N, d_2)
258     cw = Coursework(N)
259     y = cw.gradient_algorithm(do_visualisation)
260     plt.plot(y)
261     plt.title(title)
262     plt.text(24, 6, 'сетка: 21x21\nN: {}\n'.format(N))
263     plt.xticks(t)
264     plt.xlabel('итерации')
265     plt.ylabel(r'$\left| M^{-1}(\varepsilon) \right|$')
266     plt.savefig(path, dpi=200)
267     plt.clf()
268
269 def draw_mu():
270     ''' Отрисовка функций принадлежности '''
271     points_count = 21
272     d1 = 0
273     for d2 in [8, 12, 16, 20]:
274         global d_1, d_2
275         d_1 = d1
276         d_2 = d2
277         mu1 = np.ndarray(points_count)
278         mu2 = np.ndarray(points_count)
279         X = np.linspace(-1, 1, points_count, dtype=np.float)
280         for i in range(points_count):
281             fx = mu_1(X[i])
282             mu1[i] = fx
283             mu2[i] = 1 - fx
284         plt.plot(X, mu1, label=r'$\mu_1(x)$, d_2$ = {}'.format(d_2))
285         plt.plot(X, mu2, label=r'$\mu_2(x)$, d_2$ = {}'.format(d_2))
286     plt.title('Сигмоидные функции принадлежности, $d_1$ = 0')
287     plt.legend()
288     plt.savefig('pics/mu.png', dpi=200)
289     plt.clf()
290
291

```

```
292 #  
293 #  
294 draw_mu()  
295 research_N()  
296 research_d2()  
297 show_convergence_of_grad_alg(20, True)  
298 show_convergence_of_grad_alg(30, True)  
299 show_convergence_of_grad_alg(40, True)
```