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 A-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 партиции 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 21x21. 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:

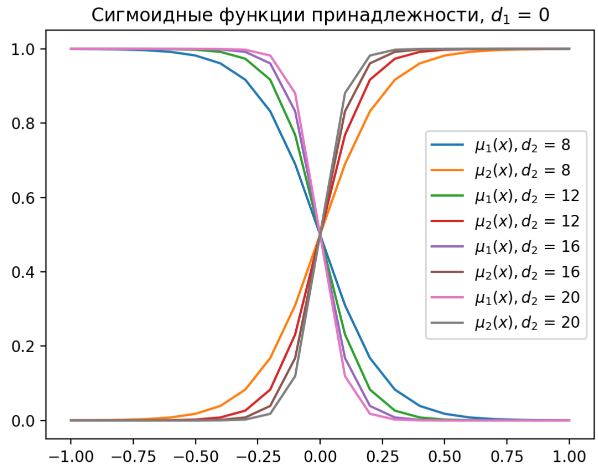
$$f^T(x) = (1, x_1, x_2, \mu_1(x_1), \mu_2(x_1), \mu_1(x_2), \mu_2(x_2),$$

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

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

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

$$\varepsilon^* = Arg \min_{\varepsilon} tr \left(M^{-1}(\varepsilon)\right)$$

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

3.3. Градиентный алгоритм

1. Выбирается невырожденный начальный план ε_N^0, s = 0.

2. Вычисляются элементы вектора градиента

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

для плана ε_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

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

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

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

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

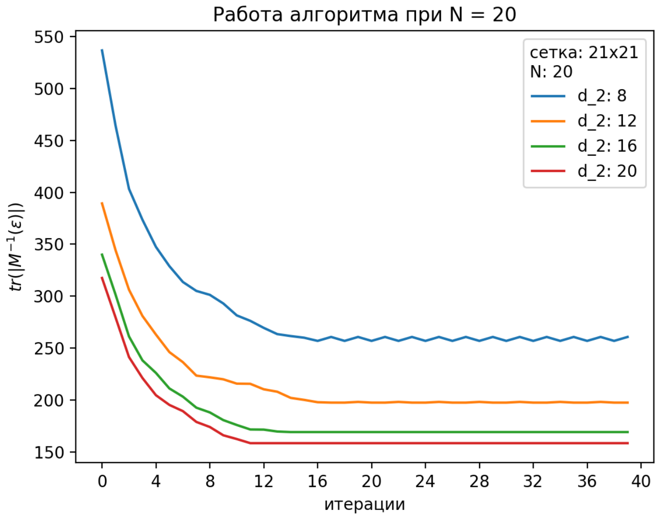
6. Сравниваются величины $\Psi\left[M\left(\varepsilon_{N,i+1}^s\right)\right]$ и $\Psi\left[M\left(\varepsilon_{N,i}^s\right)\right]$

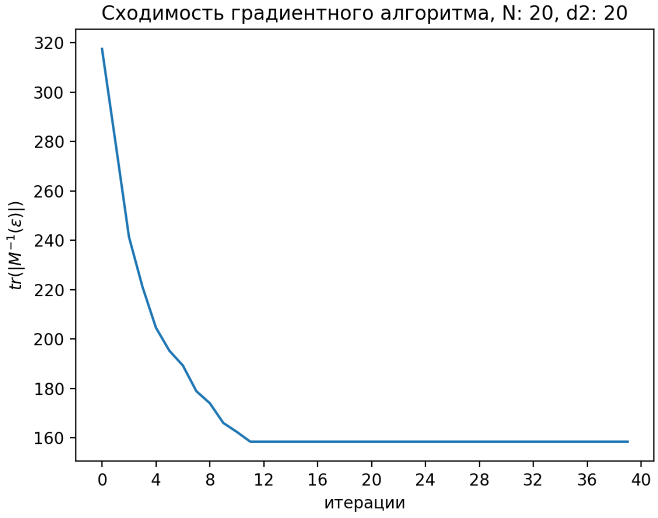
а) если $\Psi\left[M\left(\varepsilon_{N,i+1}^s\right)\right] > \Psi\left[M\left(\varepsilon_{N,i}^s\right)\right]$, то счётчик i проведённых удачных замен точек увеличивается на единицу и осуществляется переход дна шаг 3, при этом точки x^{**} и x^* исключаются из рассмотрения;

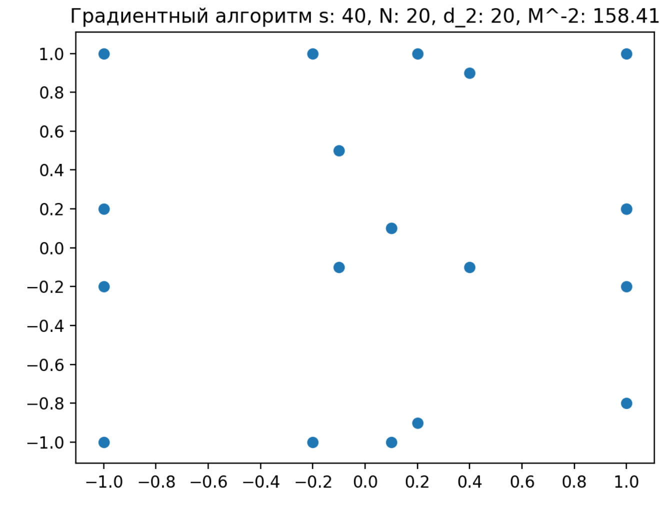
б) в противном случае: если i = 0, то вычисления прекращаются, иначе – s заменяется на s+1 и осуществляется переход на шаг 2.

4. Исследование влияния числа точек плана N

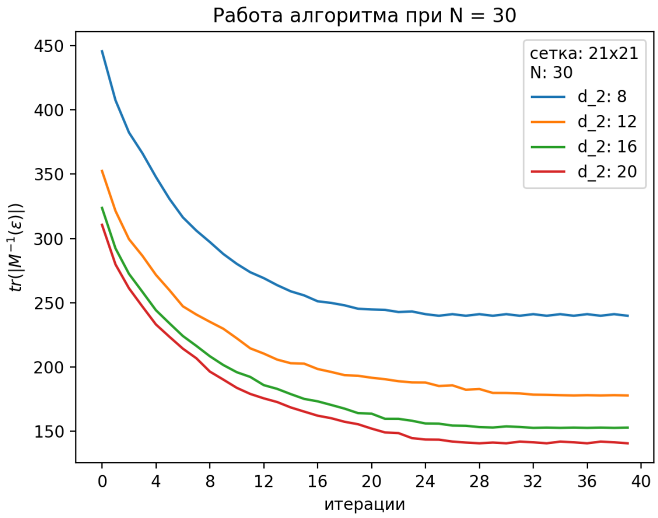
4.1. N = 20

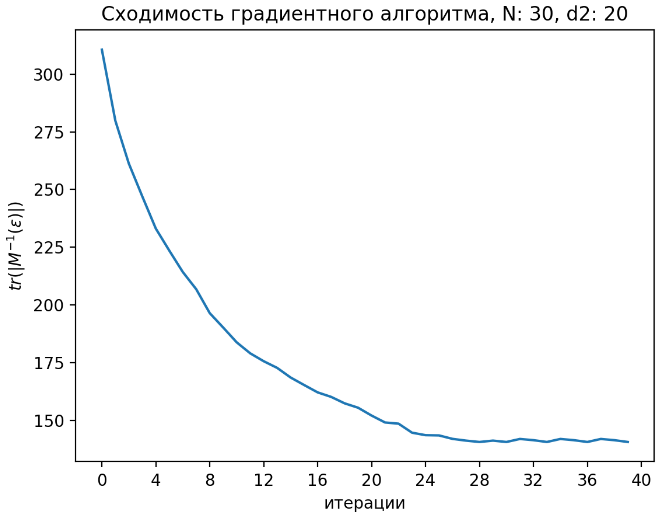


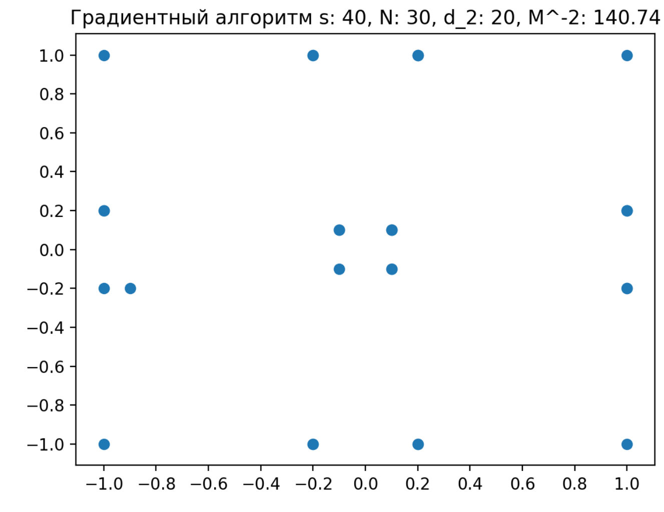




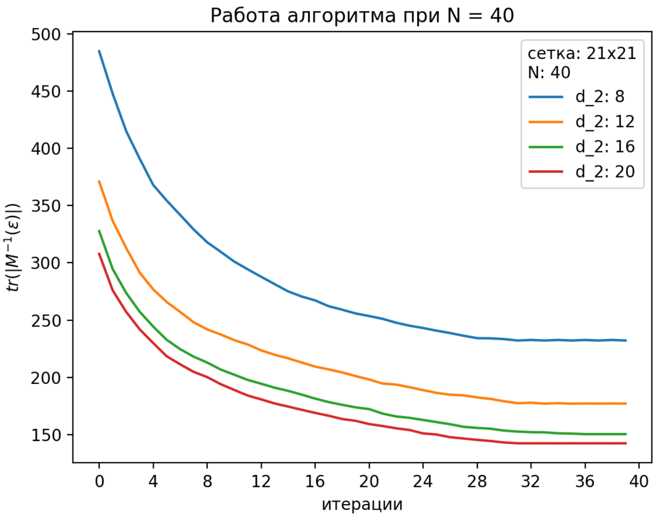
4.2. N = 30

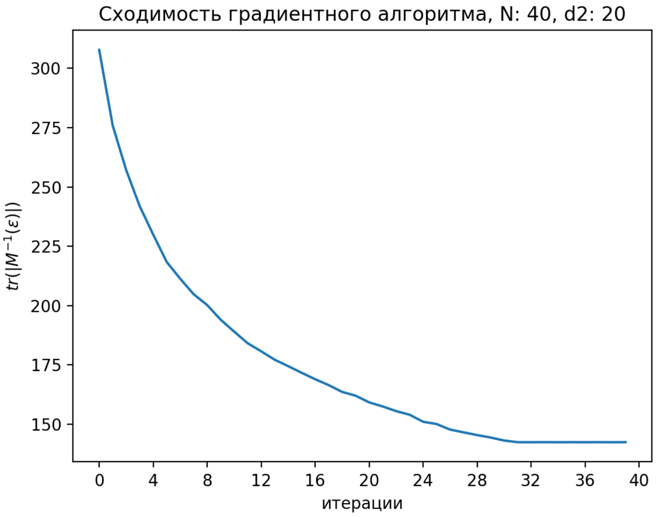

Работа алгоритма при N = 30
сетка: 21x21
N: 30
d_2: 8
d_2: 12
d_2: 16
d_2: 20
$tr(|M^{-1}(\varepsilon)|)$
итерации

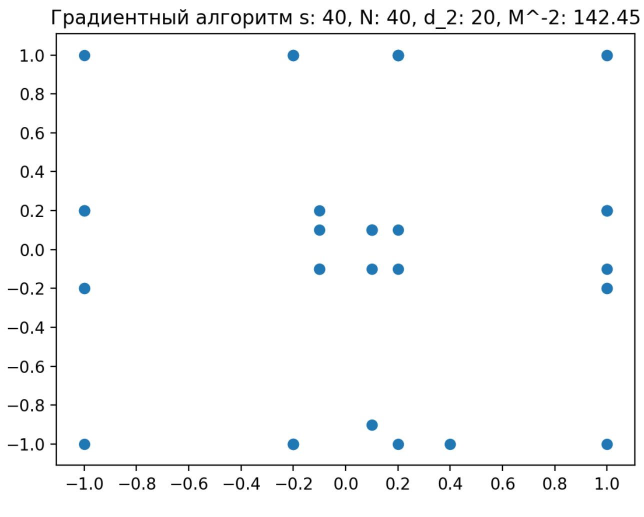

Сходимость градиентного алгоритма, N: 30, d2: 20
$tr(|M^{-1}(\varepsilon)|)$
итерации


Градиентный алгоритм s: 40, N: 30, d_2: 20, M^-2: 140.74

4.3. N = 40


Работа алгоритма при N = 40
сетка: 21x21
N: 40
d_2: 8
d_2: 12
d_2: 16
d_2: 20
$tr(|M^{-1}(\varepsilon)|)$
итерации


Сходимость градиентного алгоритма, N: 40, d2: 20
$tr(|M^{-1}(\varepsilon)|)$
итерации

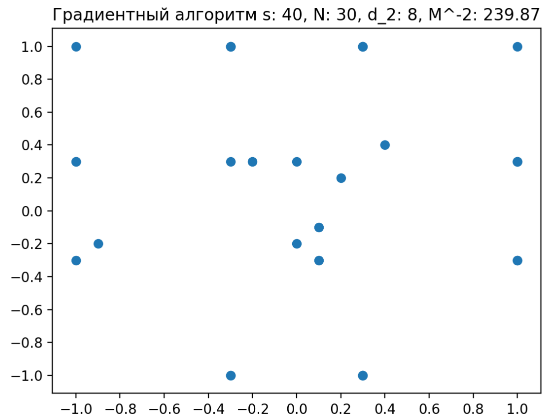

Градиентный алгоритм s: 40, N: 40, d_2: 20, M^-2: 142.45

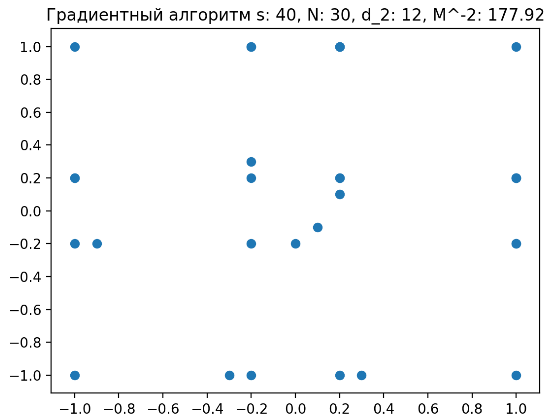
4.4. Вывод

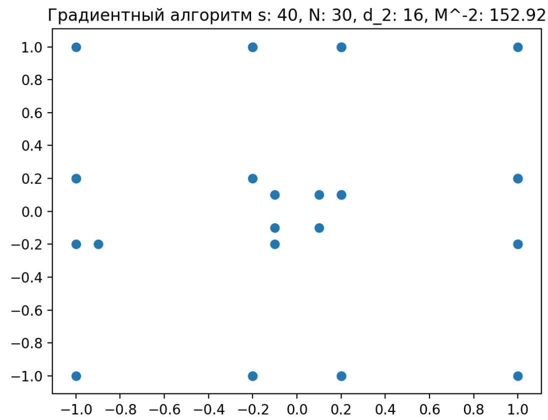
С ростом числа точек плана от 20 до 40 план становится более А-оптимальным. На 30 итерации $tr(M^{-1})$ значение падает со 160 до 140.

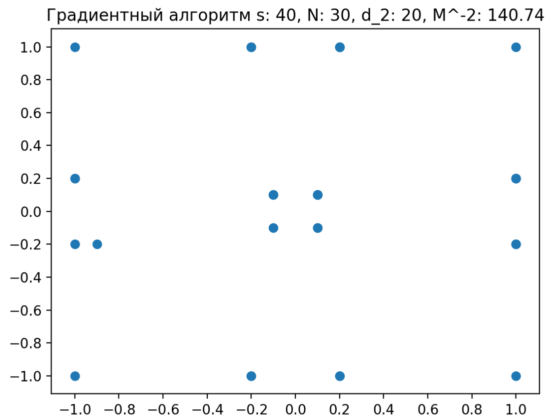
Можно заметить, что остались точки, с координатами близкими к -1, 0, 1.

5. Исседование 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 програмы

main.py

```
import matplotlib.pyplot as plt
import numpy as np
from numpy.linalg import det, inv, norm


k = 2        # число переменных
m = 9        # число параметров
d_1 = 0.0
d_2 = 20.0
MAX_ITER = 40
t = np.linspace(0, MAX_ITER, 11)


# сигмноидная функция принадлежности
def mu_1(x):
    return 1.0 - (1.0 / (1.0 + np.exp(-d_2*(x-d_1))))

def f_vector(x):
    x1 = x[0]
    x2 = x[1]
    return np.array([
        [1],
        [x1],
        [x2],
        [mu_1(x1)],
        [mu_1(x2)],
        [mu_1(x1)*x1],
        [mu_1(x2)*x1],
        [mu_1(x1)*x2],
        [mu_1(x2)*x2]
    ])

def f_vector_T(x):
    x1 = x[0]
    x2 = x[1]
    return np.array([
        1,
        x1,
        x2,
        mu_1(x1),
        mu_1(x2),
        mu_1(x1)*x1,
        mu_1(x2)*x1,
        mu_1(x1)*x2,
        mu_1(x2)*x2
    ])


#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
class Coursework():
    def __init__(self, N):
        ''' Выделение памяти под массивы '''
        width = 21
        n = width**2
        self.x_grid = np.ndarray((n, k))
```

```python
        self.x_plan = np.ndarray((N, k))
        self.M = np.ndarray((m, m))
        self.D = np.ndarray((m, m))
        self.width = width
        self.n = n
        self.N = N

    def draw_plan_on_s(self, s, A):
        alg_name = 'Градиентный алгоритм s: {}, N: {}, d_2: {:.0f}, M^−2: {:.2f}'.format(s, self.N, d_2, A)
        path = 'pics/plan_grad_alg_N_{}_s_{}_d2_{:.0f}.png'.format(self.N, s, d_2)
        t = np.linspace(−1, 1, 11)
        plt.title(alg_name)
        plt.scatter([self.x_plan[i][0] for i in range(len(self.x_plan))],[self.x_plan[i][1] for i in range(len(self.x_plan))], )
        plt.xticks(t)
        plt.yticks(t)
        plt.savefig(path, dpi=200)
        plt.clf()

    def generate_initial_guess(self):
        ''' Задаём начальное приближение '''
        # создаём сетку
        t = np.linspace(−1, 1, self.width)
        i = 0
        for x1 in t:
            for x2 in t:
                self.x_grid[i] = np.array([x1, x2])
                i+=1

        # случайно выбираем точки плана и сохраняем
        # for i in range(self.N):
        #     self.x_plan[i] = self.x_grid[np.random.choice(self.n)]
        # np.savetxt('plans/plan_{}x{}_{}.txt'.format(self.width, self.width, self.N), self.x_plan)

        # или же загружаем
        self.x_plan = np.loadtxt('plans/plan_{}x{}_{}.txt'.format(self.width, self.width, self.N), dtype=np.float)

    def build_matrix_M(self):
        ''' Построение информационной матрицы M без использования весов плана'''
        self.M = np.zeros((m, m))
        for i in range(self.N):
            x = self.x_plan[i]
            self.M += f_vector(x) * f_vector_T(x)
        self.M /= self.N

    def build_matrix_D(self):
        ''' Построение дисперсионной матрицы D '''
        self.D = inv(self.M)

    def calc_A(self):
        '''
        Критерий A − оптимальности. (A − average variance)
        Эллипсоид рассеивания с наименьшей суммой квадратов длин осей
        '''
        return np.trace(self.D)

```

```python
    def dPsi(self):
        ''' d Psi(M) / d M '''
        return np.transpose(self.D @ self.D)

#
#
    def gradient_algorithm(self, do_visualisation = False):
        '''
        Градиентный алгоритмы синтеза дискретного
        оптимального плана эксперимента
        '''
        # шаг 1
        # задаём начальное приближение
        self.generate_initial_guess()
        do_calc = True
        s = 0
        A_prev = 0.0
        A = 0.0
        result = np.ndarray(MAX_ITER)

        while do_calc == True and s < MAX_ITER:
            # шаг 2
            # вычисляем элементы вектора градиента
            self.build_matrix_M()
            self.build_matrix_D()

            grad_plan = np.ndarray(self.N)
            for i, x in enumerate(self.x_plan):
                grad_plan[i] = f_vector_T(x) @ self.dPsi() @ f_vector(x)

            grad_grid = np.ndarray(self.n)
            for i, x in enumerate(self.x_grid):
                grad_grid[i] = f_vector_T(x) @ self.dPsi() @ f_vector(x)

            i_counter = 0 # счётчик успешных замен
            do_replaces = True
            N_indicies = {i for i in range(self.N)}
            n_indicies = {i for i in range(self.n)}


            while do_replaces and i_counter < self.N:
                grad_grid = np.ndarray(self.n)
                for i, x in enumerate(self.x_grid):
                    grad_grid[i] = f_vector_T(x) @ self.dPsi() @ f_vector(x)

                # шаг 3
                # поиск x*
                max_j = 0
                max_val = grad_grid[max_j]
                for j in n_indicies:
                    if grad_grid[j] > max_val:
                        max_val = grad_grid[j]
                        max_j = j

                # шаг 4
                # поиск x**
                min_i = 0
                min_val = grad_plan[min_i]
                for i in N_indicies:
                    if grad_plan[i] < min_val:
```

```
                        min_val = grad_plan[i]
                        min_i = i

                # шаг 5
                # замена x** на x*
                self.x_plan[min_i] = self.x_grid[max_j]

                self.build_matrix_M()
                self.build_matrix_D()
                A = self.calc_A()

                # шаг 6
                # сравниваем функционал на x2- итерациях
                if A > A_prev:
                    i_counter += 1
                    N_indicies.remove(min_i)
                    n_indicies.remove(max_j)
                else:
                    do_replaces = False
                A_prev = A

            result[s] = A

            if s % 10 == 0 and do_visualisation:
                self.draw_plan_on_s(s, A)
            s += 1
            print('{}   det(M^-2): {:.2f}'.format(s, A))

        if do_calc == False:
            for i in range(s, MAX_ITER):
                result[i] = result[s-1]

        if do_visualisation:
            self.draw_plan_on_s(s, A)
        return result


#------------------------------------------------------------------------------
#------------------------------------------------------------------------------
def perform_experiment(N, do_visualisation = False):
    ''' Отдельный эксперимент '''
    cw = Coursework(N)
    return cw.gradient_algorithm(do_visualisation)

def research_N():
    ''' Исследование работы алгоритма при различных N '''
    print('Исследование работы алгоритма при различных N')
    d1 = 0
    for N in [20, 30, 40]:
        for d2 in [8, 12, 16, 20]:
            global d_1, d_2
            d_1 = float(d1)
            d_2 = float(d2)
            y = perform_experiment(N, False)
            plt.plot(y, label='d_2: {:.0f}'.format(d2))
        plt.title('Работа алгоритма при N = {}'.format(N))
        plt.legend(title='сетка: 21x21\nN: {}'.format(N))
        plt.xticks(t)
        plt.xlabel('итерации')
        plt.ylabel(r'$tr(\left| M^{-1}(\varepsilon) \right|)$')
```

```
        plt.savefig('pics/research_N_{}.png'.format(N), dpi=200)
        plt.clf()

def research_d2():
    ''' Исследование работы алгоритма при различных d2 '''
    print('Исследование работы алгоритма при различных d2')
    N = 30
    d1 = 0
    for d2 in [8, 12, 16, 20]:
        global d_1, d_2
        d_1 = float(d1)
        d_2 = float(d2)
        y = perform_experiment(N, True)
        plt.plot(y, label='d_2: {:.0f}'.format(d2))
    plt.title(r'Работа алгоритма при N = {}'.format(N))
    plt.legend(title='сетка: 21x21\nN: {}'.format(N))
    plt.xticks(t)
    plt.xlabel('итерации')
    plt.ylabel(r'$tr(\left| M^{-1}(\varepsilon) \right|)$')
    plt.savefig('pics/research_d2_N_{}.png'.format(N), dpi=200)
    plt.clf()

def show_convergence_of_grad_alg(N, do_visualisation = False):
    ''' Отрисовка сходимости градиентного алгоритма '''
    print('Отрисовка сходимости градиентного алгоритма')
    title = 'Сходимость градиентного алгоритма, N: {}, d2: {:.0f}'.format(N, d_2)
    path = 'pics/convergence_grad_alg_N_{}_d2_{:.0f}.png'.format(N, d_2)
    cw = Coursework(N)
    y = cw.gradient_algorithm(do_visualisation)
    plt.plot(y)
    plt.title(title)
    plt.text(24, 6, 'сетка: 21x21\nN: {}\n'.format(N))
    plt.xticks(t)
    plt.xlabel('итерации')
    plt.ylabel(r'$tr(\left| M^{-1}(\varepsilon) \right|)$')
    plt.savefig(path, dpi=200)
    plt.clf()

def draw_mu():
    ''' Отрисовка функций принадлежности  '''
    points_count = 21
    d1 = 0
    for d2 in [8, 12, 16, 20]:
        global d_1, d_2
        d_1 = d1
        d_2 = d2
        mu1 = np.ndarray(points_count)
        mu2 = np.ndarray(points_count)
        X = np.linspace(-1, 1, points_count, dtype=np.float)
        for i in range(points_count):
            fx = mu_1(X[i])
            mu1[i] = fx
            mu2[i] = 1 - fx
        plt.plot(X, mu1, label=r'$\mu_1(x), d_2$ = {}'.format(d_2))
        plt.plot(X, mu2, label=r'$\mu_2(x), d_2$ = {}'.format(d_2))
    plt.title('Сигмоидные функции принадлежности, $d_1$ = 0')
    plt.legend()
    plt.savefig('pics/mu.png', dpi=200)
    plt.clf()

```

```
#----------------------------------------------------------------------------------------------------------------------
#----------------------------------------------------------------------------------------------------------------------
draw_mu()
research_N()
research_d2()
show_convergence_of_grad_alg(20, True)
show_convergence_of_grad_alg(30, True)
show_convergence_of_grad_alg(40, True)
```