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 непрерывных A-оптимальных планов эксперимента для нечетких квадратичных однофакторных моделей с двумя подобластями определения»

Факультет: ФПМИ
Группа: ПМ-63
Студент: Кожекин М.В.
Вариант: 5

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 непрерывных A-оптимальных планов эксперимента для однофакторных моделей.

3. Работа приложения должна быть продемонстрирована на нескольких тестовых примерах. Оптимальность полученных планов должна быть подтверждена выполнением соответствующих условий с приемлемой точностью.

2. Требования

Приложение должно осуществлять синтез непрерывных A-оптимальных планов эксперимента для нечетких регрессионных моделей с одним вещественным фактором.

Область определения вещественной переменной (интервал [–1; +1]) при фаззификации разбивается на две трапецевидные нечеткие партиции с функциями принадлежности:

$$\mu_1(x) = \begin{cases} 1, & x \le -\Delta \\ \frac{\Delta - x}{2\Delta}, & -\Delta \le x \le \Delta \\ 0, & x \ge \Delta \end{cases}$$

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

Построить оптимальные планы для значений $\Delta = 0.5; 0.4; 0.3; 0.2$. Базовая модель квадратичная. Характеристики построенных планов представить в таблице, координаты точек спектра планов отобразить на рисунке вместе с функциями принадлежности.

3. Анализ

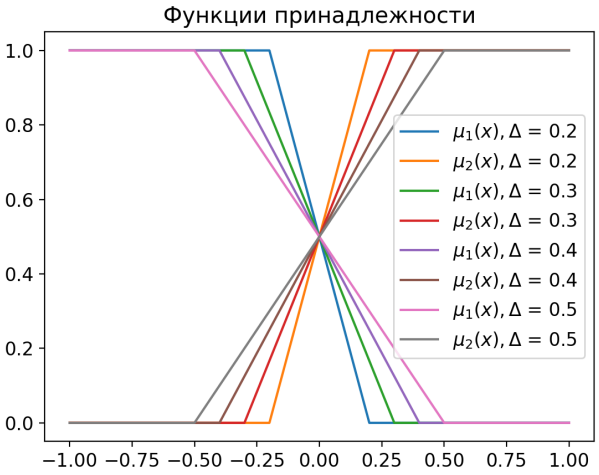
3.1. Нечёткая логика

В исходном виде квадратичная однофакторная модель по своему списку регрессоров имеет вид:

$$f^T(x) = (1, x_1, x_1^2, \mu_1(x_1), \mu_2(x_1), \mu_1(x_1)x_1, \mu_2(x_1)x_1, \mu_1(x_1)x_1^2, \mu_2(x_1)x_1^2)$$

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

$$f^T(x) = (1, x_1, x_1^2, \mu_1(x_1), \mu_1(x_1)x_1, \mu_1(x_1)x_1^2$$



3.2. Последовательный алгоритм

1. Выбирается невырожденный начальный план ε^0. Номер итерации s = 0.

2. Отыскивается точка глобального эстремума x^s:

$$x^s = arg\min_{x\in\tilde{X}} \varphi(x,\varepsilon^s), \text{ где } \quad \varphi(x,\varepsilon) = f^T(x)\frac{d\Phi[M(\varepsilon)]}{dM(\varepsilon)}f(x)$$

3. Проверяется приближенное выполнение необходимых и достаточных условий оптимальности планов

$$\left|-\min_{x\in\tilde{X}}\varphi(x,\varepsilon^s) + trM(\varepsilon^s)\frac{d\Phi[M(\varepsilon)]}{dM(\varepsilon)}\right| \le \delta, \text{ где } \quad \delta = \left|\min_{x\in\tilde{X}}\varphi(x,\varepsilon^s)\right| \times 0.01$$

Если условие выполнено, то работа алгоритма прекращается. В противном случае осуществляется переход на шаг 4.

4. Составляется план

$$\varepsilon^{s+1} = (1-\alpha^s)\varepsilon^s + \alpha^s\varepsilon(x^s)$$

где $\alpha \in (0,1)$, $\varepsilon(x^s)$ - план, состоящий из одной точки x^s.

5. Величина $\Psi[M(\varepsilon^{s+1})]$ сравнивается с величиной $\Psi[M(\varepsilon^s)]$:

а) если $\Psi[M(\varepsilon^{s+1})] \ge \Psi[M(\varepsilon^s)]$, то α^s уменьшается в γ раз и повторяются шаги 4-5;

б) если имеет место обратное неравенство, то s = s+1 и происходит переход на шаг 2.

В конце происходит процедура "очистки" плана.

1. Точки, тяготеющие к одной из групп, объединяются по правилу

$$p_j^s = \sum_{k=1}^{l} p_{j_k}^s, \quad x_j^s = \frac{\sum_{k=1}^{l} x_{j_k}^s p_{j_k}^s}{p_j^s}$$

2. Точки с малыми весами, не тяготеющие ни к одной из групп, указанных в 4), выбрасываются. Их веса распределяются между остальными точками

3.3. Критерий A-оптимальности и проиводная

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

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

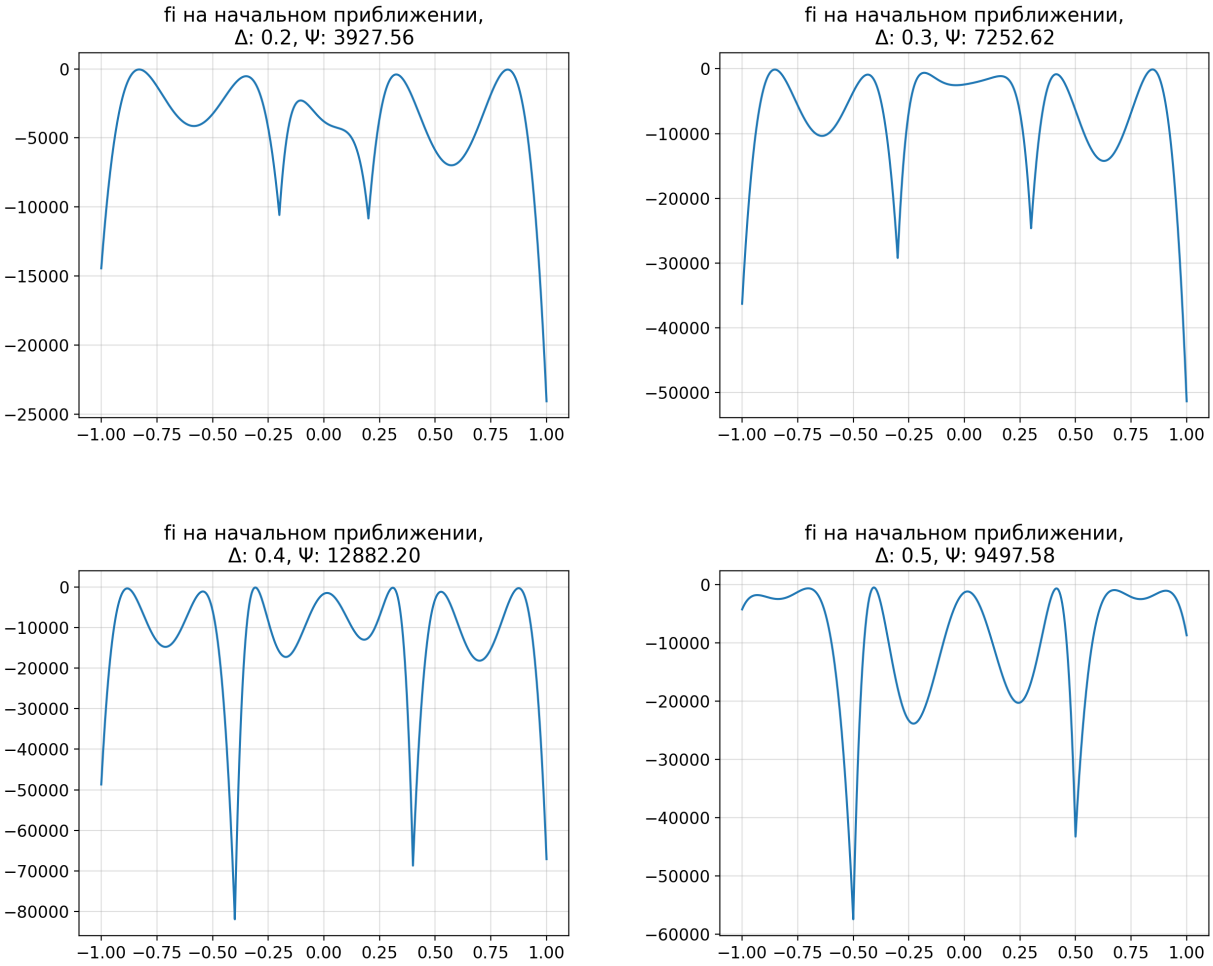
4. Работа алгоритма

4.1. Параметры начального приближения и алгоритма

Точки спектра плана находятся в интервале [-1, +1] с шагом 0.004. Всего у нас их 501 штука, вес каждой равен $1/501$. Максимальное число итераций по s $= 200$, максимальное число итераций по $\alpha = 30$.

4.2. Асимметрия функции fi

Для начала рассмотрим значения функции $\varphi(x, \varepsilon^s)$ на начальной итерации. Как мы видим, для каждого значения Δ функция асимметрична. Это может быть вызвано ошибками численного вычисления производной по функционалу. Символьное вычисление могло бы исправить эту ситуацию.

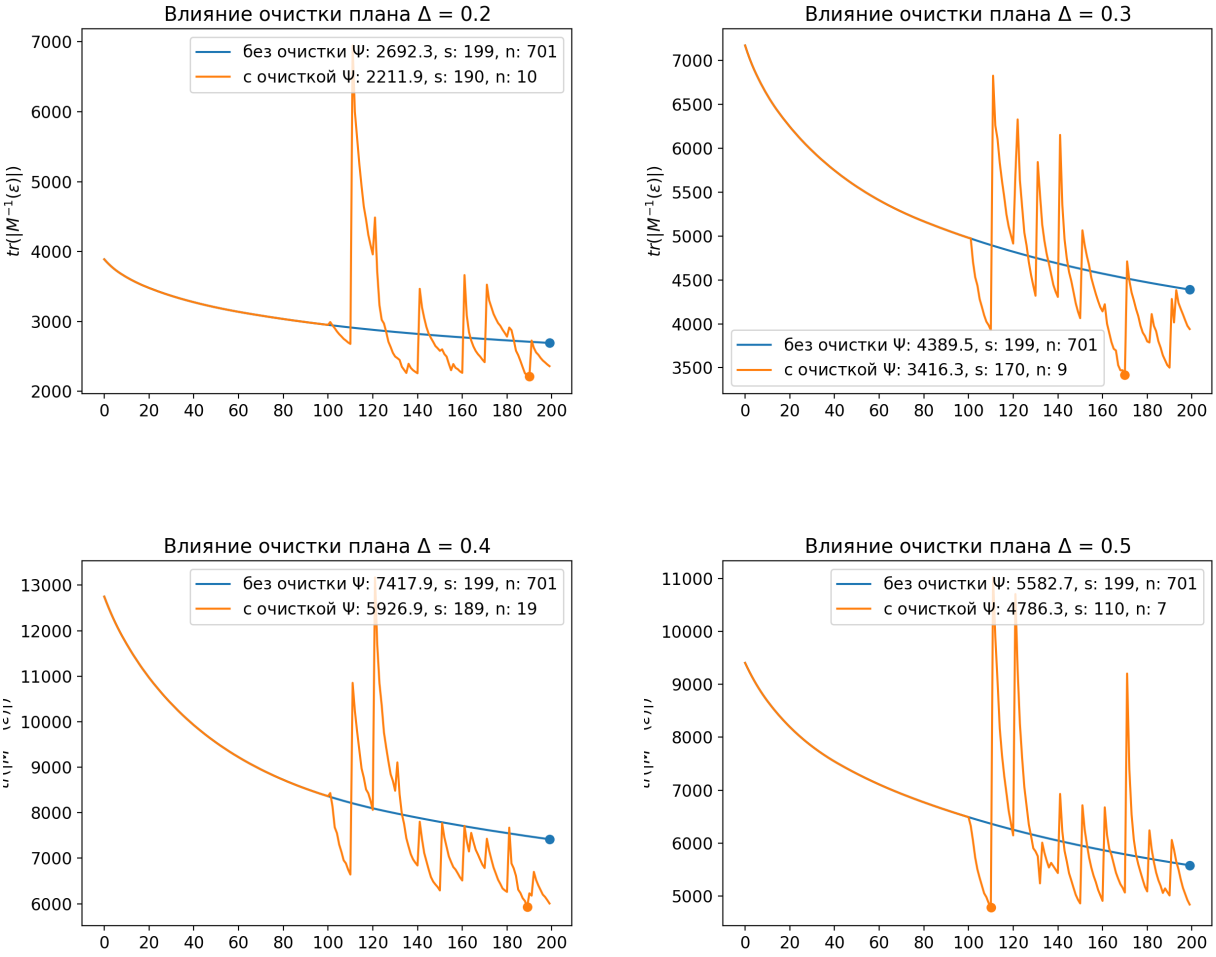


4.3. Очистка плана

В данном алгоритме мы производим очистку плана в 3 этапа:

1. Объединение повторяющихся точек.
2. Удаление малозначительных ($p_i < 0.00009$) точек и пересчёт весов.
3. Объединение точек в радиусе 0.1 используя центр масс

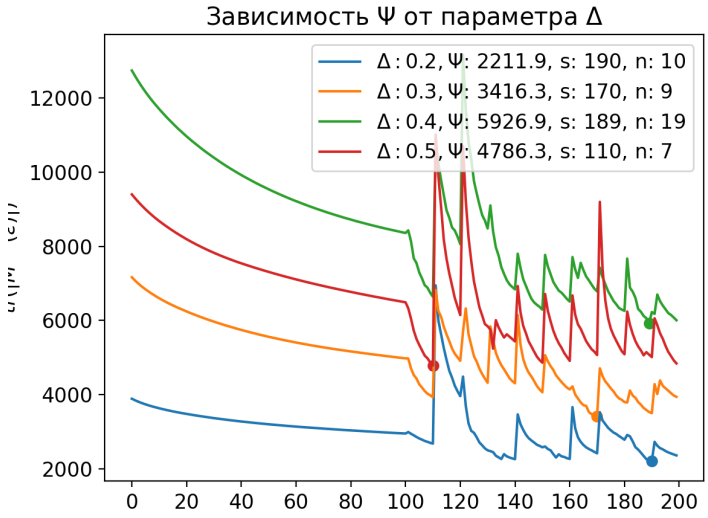
Было проведено 8 эксперименов: 4 с очисткой плана и 4 без. Очистка проиводилась каждые 10 шагов алгоритма, начиная с 100-й итерации.



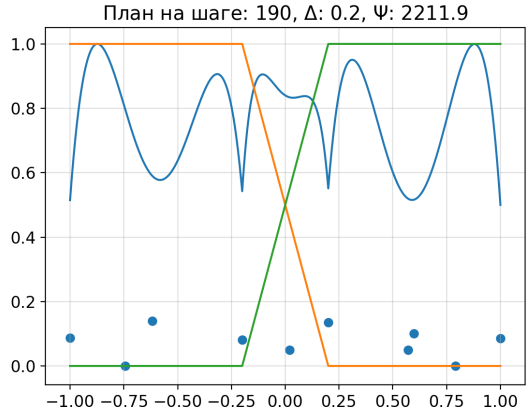
Вывод:

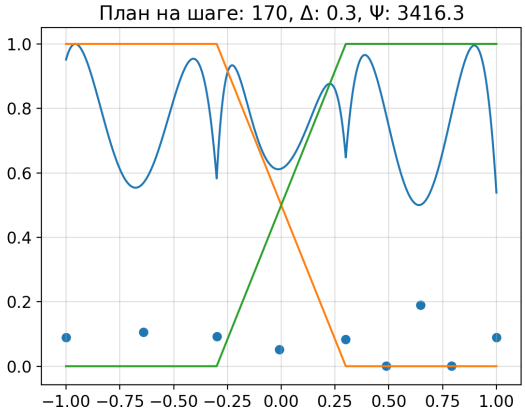
Алгоритм без очистки плана сходится плавно, но медленно. Очистка спектра плана позволяет найти на 15-30% более A-оптимальный план, имеющий в 35-70 раз меньше точек. Также это позволяет ускорить вычисления.

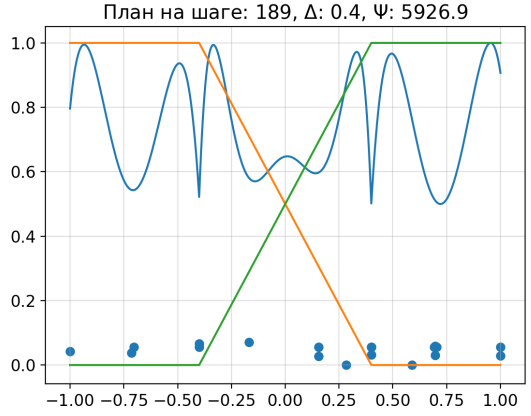
5. Исследование влияния параметра Δ

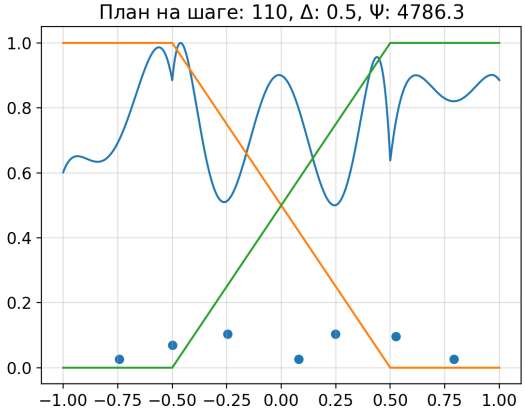


Как видно из графиков наиболее оптимальным значением параметра Δ является 0.2. При нём функционал $\Psi \approx 2200$ хотя при $Delta = 0.4$ он в 2.5 раза больше $\Psi \approx 5900$. Получившиеся в результате работы алгоритма планы:









6. Исходный код программы

main.py

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

np.set_printoptions(precision=2, suppress=True)
plt.rcParams.update({'font.size': 12})


m = 6                    # число параметров
n = 501                  # число точек плана
grid_width = 501         # число точек сетки
Delta = 0.1
eps_close = 0.1
MAX_ITER = 200
t = np.linspace(0, MAX_ITER, 11)
t_full = np.arange(MAX_ITER)


def mu_1(x):
    if x <= -Delta:
        return 1
    elif -Delta <= x and x <= Delta:
        return (Delta - x) / (2.0*Delta)
    else:
        return 0

def f_vector(x):
    return np.array([[1], [x], [x**2], [mu_1(x)], [mu_1(x)*x], [mu_1(x)*x**2]])

def f_vector_T(x):
    return np.array([ 1, x, x**2, mu_1(x), mu_1(x)*x, mu_1(x)*x**2 ])


#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
class Coursework():
    def __init__(self, do_visualisation = False, do_clear = False):
        ''' Выделение памяти под массивы '''
        self.x = np.ndarray(n)
        self.p = np.ndarray(n)
        self.grid = np.linspace(-1, 1, grid_width)
        self.M = np.ndarray((m, m))
        self.D = np.ndarray((m, m))
        self.alpha = 1.0 / n
        self.gamma = 2.0
        self.max_iter_s = MAX_ITER
        self.max_iter_alpha = 30
        self.do_clear = do_clear
        self.do_visualisation = do_visualisation

    def generate_initial_guess(self):
        ''' Задаём начальное приближение '''
        self.x = np.linspace(-1, 1, grid_width)
        self.p = np.full(n, 1.0 / n)
```

```
57
58      def fi(self, x):
59          ''' Значение функции fi в точке x '''
60          return f_vector_T(x) @ self.dPsi() @ f_vector(x)
61
62      def calc_min_fi_and_point(self):
63          ''' Поиск максимального значения fi и точки '''
64          min_fi_point = self.grid[0]
65          min_fi = self.fi(min_fi_point)
66          for point in self.grid:
67              fi = self.fi(point)
68              if fi < min_fi:
69                  min_fi = fi
70                  min_fi_point = point
71          return min_fi, min_fi_point
72
73      def is_plan_approximately_optimal(self, min_fi):
74          '''
75          Проверяем приближенное выполнение необходимых и
76          достаточных условий оптимальности плана
77          '''
78          delta = 0.01 * abs(min_fi)
79          if abs(-min_fi + np.trace(self.M @ self.dPsi())) <= delta:
80              return True
81          else:
82              return False
83
84      def check_necessity_and_sufficiecy(self):
85          '''
86          Проверяем необходимые и достаточные условий оптимальности
87          '''
88          right_part = self.calc_A()   # tr M^{-1}(e*)
89
90          x_ = self.x[0]
91          dPsi = self.dPsi()
92          max_val = np.trace(-dPsi @ (f_vector(x_) * f_vector_T(x_)))
93
94          for x in self.x:
95              val = np.trace(-dPsi @ (f_vector(x) * f_vector_T(x)))
96              if val > max_val:
97                  x_ = x
98                  max_val = val
99
100         left_part = max_val
101         return left_part, right_part
102
103     def clear_plan(self):
104         ''' Процедура очистки плана '''
105         global n
106         p = np.zeros(grid_width)
107
108         # убираем совпадающие точки
109         for plan_point, weight in zip(self.x, self.p):
110             for i, grid_point in enumerate(self.grid):
111                 if abs(plan_point - grid_point) < 1e-8:
112                     p[i] += weight
113         self.x = np.copy(self.grid)
114         self.p = np.copy(p)
115         n = len(self.x)
116
```

```
        # убираем незначительные точки
        p_weigth = 0.0
        i = 0
        x_new = np.array([])
        p_new = np.array([])
        for point, weight in zip(self.x, self.p):
            if weight < 0.00009:
                p_weigth += weight
            else:
                x_new = np.append(x_new, [point])
                p_new = np.append(p_new, [weight])
                i+=1
        x_new.resize(i)
        p_new.resize(i)
        n = i
        sum_p = np.sum(self.p)
        self.x = np.copy(x_new)
        self.p = np.copy(p_new)
        self.p /= (1.0 - p_weigth / sum_p)

        # объединяем точки, используя центр масс
        x_new = []
        p_new = []
        indicies = {i for i in range(len(self.x))}
        while len(indicies) > 0:
            point = self.x[list(indicies)[0]]
            indicies_tmp = copy.deepcopy(indicies)
            p_sum = 0.0
            xp_sum = 0.0
            for j in indicies_tmp:
                close_point = self.x[j]
                if abs(point - close_point) < eps_close :
                    p_sum += self.p[j]
                    xp_sum += self.x[j] * self.p[j]
                    indicies.remove(j)
            x_s = xp_sum / p_sum
            p_s = p_sum
            x_new += [x_s]
            p_new += [p_s]
        self.x = np.array(x_new)
        self.p = np.array(p_new)
        n = len(self.x)

    def add_new_point(self, x_s):
        ''' Добавляем в план новую точку x_s '''
        global n
        n += 1
        self.x = np.append(self.x, [x_s], axis=0)
        self.p = np.append(self.p * (1.0 - self.alpha), self.alpha)

    def draw_plan_on_s(self, s, Psi):
        ''' Отрисовка весов плана эксперимента '''
        mu1 = np.ndarray(grid_width)
        mu2 = np.ndarray(grid_width)
        X = np.copy(self.grid)
        Fi = np.copy(self.grid)
        for i in range(grid_width):
            fx = mu_1(X[i])
            Fi[i] = self.fi(X[i])
            mu1[i] = fx
```

```
            mu2[i] = 1 - fx

        # нормализуем
        min_Fi = np.min(Fi)
        max_Fi = np.max(Fi)
        Fi = Fi - min_Fi
        Fi = Fi / (2*(max_Fi - min_Fi))  + 0.5

        plt.plot(X, Fi)
        plt.plot(X, mu1)
        plt.plot(X, mu2)
        plt.scatter(self.x, self.p)

        plt.title('План на шаге: {}, $\Delta$: {:.1f}, $\Psi$: {:.1f}'.format(s,
    Delta, Psi))
        plt.grid(alpha=0.4)
        plt.margins(0.05)
        if self.do_clear:
            plt.savefig('pics/plan_delta_clear_{}_s_{}.png'.format(Delta, s),
    dpi=200)
        else:
            plt.savefig('pics/plan_delta_{}_s_{}.png'.format(Delta, s), dpi=200)
        plt.clf()

    def draw_fi_init(self, Psi):
        ''' Отрисовка функции Fi на начальном приближении '''
        X = np.copy(self.grid)
        Fi = np.copy(self.grid)
        for i in range(grid_width):
            Fi[i] = self.fi(X[i])

        plt.plot(X, Fi)
        plt.title('fi на начальном приближении,\n$\Delta$: {:.1f}, $\Psi$: {:.2f}'
    .format(Delta, Psi))
        plt.grid(alpha=0.4)
        plt.margins(0.05)
        plt.savefig('pics/fi_delta_{:.1f}.png'.format(Delta), dpi=200)
        plt.clf()

    def build_table(self, s):
        ''' Построение таблицы плана на итерации '''
        fi_values = np.ndarray(n)
        for i, x in enumerate(self.x):
            fi_values[i] = self.fi(x)

    def build_matrix_M(self):
        ''' Построение информационной матрицы M '''
        self.M = np.zeros((m, m))
        for i, x in enumerate(self.x):
            self.M += self.p[i] * f_vector(x) * f_vector_T(x)

    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)

    def dPsi(self):
        ''' d Psi(M) / d M '''
        return -matrix_power(self.M, -2)


#-------------------------------------------------------------------------------
#-------------------------------------------------------------------------------
    def sequential_algorithm(self):
        '''
        Последовательный алгоритм синтеза непрерывного
        оптимального плана эксперимента
        '''
        # 1 этап
        # задаём начальное приближение
        self.generate_initial_guess()
        do_calc = True
        s = 0
        result = np.ndarray(self.max_iter_s)
        l = np.ndarray(self.max_iter_s)
        r = np.ndarray(self.max_iter_s)
        points_count = np.ndarray(self.max_iter_s)
        self.build_matrix_M()
        self.build_matrix_D()
        Psi = self.calc_A()
        self.min_s = s
        self.min_Psi = Psi
        self.draw_fi_init(Psi)


        while do_calc == True and s < self.max_iter_s:
            a = 0
            self.alpha = 1.0 / n
            Psi_prev = Psi

            # 2 этап
            # поиск точки глобального экстремума x_s
            min_fi, x_s = self.calc_min_fi_and_point()

            # 3 этап
            # приближенное выполнение необходимых и достаточных
            # условий оптимальности плана
            do_calc = not self.is_plan_approximately_optimal(min_fi)

            # 4 этап
            # добавление точки x_s в спектр плана
            self.add_new_point(x_s)
            self.build_matrix_M()
            self.build_matrix_D()
            Psi = self.calc_A()

            # 5 этап
            # Уменьшаем шаг, если метод расходится
            while Psi >= Psi_prev and a < self.max_iter_alpha:
                a += 1
                self.alpha /= self.gamma
                self.add_new_point(x_s)
                self.build_matrix_M()
                self.build_matrix_D()
```

```
                Psi_prev = Psi
                Psi = self.calc_A()

            if self.do_clear and s % 10 == 0 and s >= 100:
                self.clear_plan()
                Psi = self.calc_A()

            # Строим таблицы и графики
            if self.do_visualisation and s % 10 == 0:
                self.draw_plan_on_s(s, Psi)

            result[s] = Psi
            points_count[s] = n
            l[s], r[s] = self.check_necessity_and_sufficiecy()


            if result[s] < self.min_Psi:
                self.min_s = s
                self.min_Psi = result[s]
                self.x_best = np.copy(self.x)
                self.p_best = np.copy(self.p)

            # print('s: {}, a: {}, x_s: {:.2f}, len(x): {}, Psi: {:.3f}, n: {}'.format(s, a, x_s, self.x.shape, Psi, n))
            s += 1

        self.x = np.copy(self.x_best)
        self.p = np.copy(self.p_best)
        self.draw_plan_on_s(self.min_s, self.min_Psi)
        print(self.min_s, self.min_Psi, len(self.x))
        return result, l, r, points_count


#-----------------------------------------------------------------------------
#-----------------------------------------------------------------------------
def draw_mu():
    ''' Отрисовка функций принадлежности  '''
    print('Отрисовка функций принадлежности')
    global Delta
    for delta in [0.2, 0.3, 0.4, 0.5]:
        Delta = delta
        mu1 = np.ndarray(grid_width)
        mu2 = np.ndarray(grid_width)
        X = np.linspace(-1, 1, grid_width)
        for i in range(grid_width):
            fx = mu_1(X[i])
            mu1[i] = fx
            mu2[i] = 1 - fx
        plt.plot(X, mu1, label=r'$\mu_1(x), \Delta$ = {:.1f}'.format(delta))
        plt.plot(X, mu2, label=r'$\mu_2(x), \Delta$ = {:.1f}'.format(delta))
    plt.title('Функции принадлежности')
    plt.legend()
    plt.savefig('pics/mu.png', dpi=200)
    plt.clf()

def perform_experiment(do_visualisation = False, do_clear = False):
    ''' Отдельный эксперимент '''
    cw = Coursework(do_visualisation, do_clear)
    return cw.sequential_algorithm()

```

```
def research_delta(do_visualisation = True, do_clear = True):
    ''' Исследование зависимости  '''
    global Delta, n
    y = np.ndarray((4, MAX_ITER))
    l = np.ndarray((4, MAX_ITER))
    r = np.ndarray((4, MAX_ITER))
    points_count = np.ndarray((4, MAX_ITER))

    print('Отрисовка сеток')
    deltas = [0.2, 0.3, 0.4, 0.5]
    for i in range(4):
        Delta = deltas[i]
        n = grid_width
        y[i], l[i], r[i], points_count[i] = perform_experiment(do_visualisation,
     do_clear)

    print('Отрисовка сходимости')
    for i in range(4):
        Delta = deltas[i]
        i_min, i_val = np.argmin(y[i]), np.min(y[i])
        n_min = int(points_count[i][i_min])
        plt.scatter(i_min, i_val)
        plt.plot(y[i], label='$\Delta: {:.1f}, \Psi$: {:.1f}, s: {}, n: {}'.
    format(Delta, i_val, i_min, n_min))
        # plt.plot(l[i])
        # plt.plot(r[i])

    plt.title(r'Зависимость $\Psi$ от параметра $\Delta$')
    plt.legend()
    plt.xticks(t)
    plt.ylabel(r'$tr(\left| M^{-1}(\varepsilon) \right|)$')
    if do_clear:
        plt.savefig('pics/convergence_delta_yes_clear.png', dpi=200)
    else:
        plt.savefig('pics/convergence_delta_no_clear.png', dpi=200)
    plt.clf()


def research_clear():
    ''' Исследование влияние очистки плана  '''
    global Delta, n
    for Delta in [0.2, 0.3, 0.4, 0.5]:
        y = np.ndarray((2, MAX_ITER))
        l = np.ndarray((2, MAX_ITER))
        r = np.ndarray((2, MAX_ITER))
        points_count = np.ndarray((2, MAX_ITER))

        n = grid_width
        y[0], l[0], r[0], points_count[0] = perform_experiment(False, False)
        n = grid_width
        y[1], l[1], r[1], points_count[1] = perform_experiment(False, True)

        i_min, i_val = np.argmin(y[0]), np.min(y[0])
        n_min = int(points_count[0][i_min])
        plt.scatter(i_min, i_val)
        plt.plot(y[0], label='без очистки $\Psi$: {:.1f}, s: {}, n: {}'.format(
    i_val, i_min, n_min))

        i_min, i_val = np.argmin(y[1]), np.min(y[1])
        n_min = int(points_count[1][i_min])
```

```
410         plt.scatter(i_min, i_val)
411         plt.plot(y[1], label='с очисткой $\Psi$: {:.1f}, s: {}, n: {}'.format(
    i_val, i_min, n_min))
412 
413         plt.title('Влияние очистки плана $\Delta$ = {:.1f}'.format(Delta))
414         plt.legend()
415         plt.xticks(t)
416         plt.ylabel(r'$tr(\left| M^{-1}(\varepsilon) \right|)$')
417         plt.savefig('pics/research_clear_{:.1f}.png'.format(Delta), dpi=200)
418         plt.clf()
419 
420 
421 #-------------------------------------------------------------------------------
422 #-------------------------------------------------------------------------------
423 draw_mu()
424 research_delta(False, False)
425 research_delta(False, True)
426 research_clear()
```